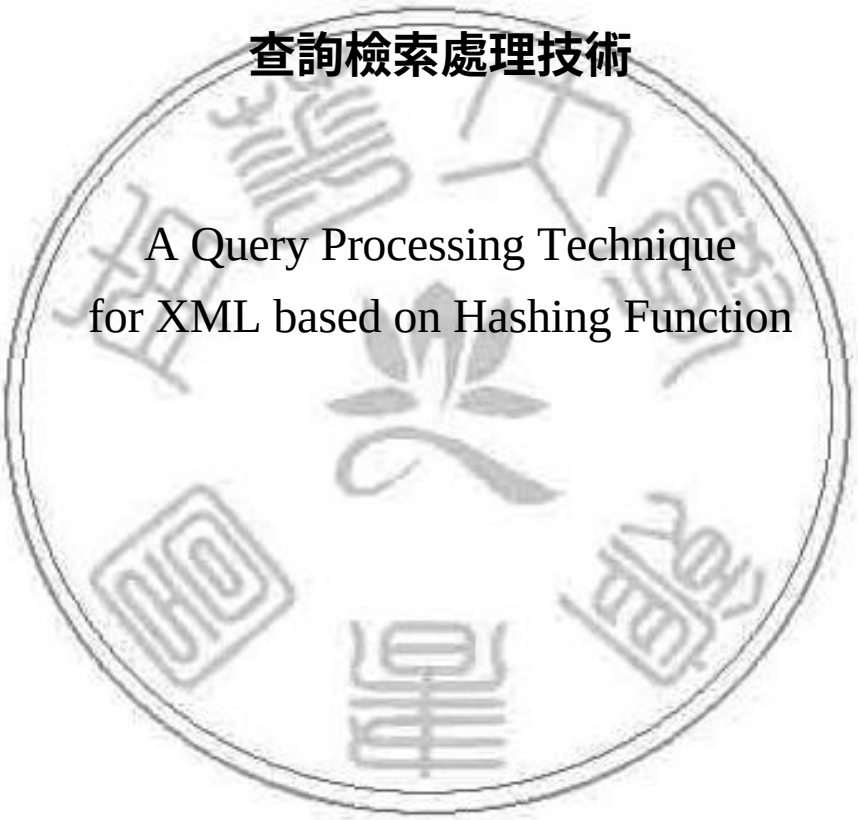
南　華　大　學

資訊管理學系碩士論文

以雜湊函數為索引編碼基礎的 XML

查詢檢索處理技術



A Query Processing Technique
for XML based on Hashing Function

研 究 生：李 建 圖

指導教授：吳 光 閔

中華民國　九 十 二 年　六 月

南　華　大　學

碩士學位論文

資訊管理研究所

以雜湊函數為索引編碼基礎的 XML 查詢檢索處理技術

A Query Processing Technique for XML based on Hashing Function

研 究 生：李建國

經考試合格特此證明

口試委員：黃士銘

王勇斗

指導教授：吳光明

所　　長：

口試日期：中華民國 92 年 6 月 13 日

以雜湊函數為索引編碼基礎的 XML 查詢檢索處理技術

學生：李建圖　　指導教授：吳光閔 博士

南華大學資訊管理學系碩士班

摘　　要

XML 制定的主要目地是為了能在網際網路上傳送交換資料或處理資訊，XML 所著重的是文件資料的結構描述，所以在資料交換或呈現方法上有著一定的特色與優勢，但隨著所採用的人愈來愈普及，相對的資料量也愈來愈龐大，衍生的問題便是資料資訊查詢的問題。本論文提出一個新的資料檢索技術，透過 XML API 的運用，利用 hashing 的技巧，將 XML DOM 上的節點加以編碼，以註記於附屬的節點中，在查尋檢索的過程中，透過提出查詢的演算法，可在 XML 資料中快速的檢索，效率極佳。在實驗中結果顯示出本論文所提出之檢索方法的效率高於 [25] 所提出之方法，平均效率約改善了 51%。

關鍵字：XML、索引、查詢、編碼。

A Query Processing Technique

for XML based on Hashing Function

Student：Chien-Tu Li　　Advisors：Dr. Guang-Ming Wu

Institute of Information Management
Nan-Hua University

ABSTRACT

The most important goal of making “XML” is for exchanging or handling the information on the Web. The emphasis of XML is the structure of document, so the transformation of information and the ways of expressing has some characters and advantages. As people using the system is more and more popular, there is more and more information relatively. The question of that is searching the information. This paper provides a new technique of searching XML documents by using hashing function and XML API to sign on DOM nodes. In the process of searching, by provides for the ways of algorithm can search the XML information quickly and it is very efficiency. The Experimental results show that our searching method is more efficiency outperform [25]. The efficiency improves about 51%.

Keywords: index, query, signature, XML.

目　錄

書名頁 …… I

碩博士論文授權書 …… II

論文指導教授推薦函 …… III

論文口試委員審定書 …… IV

中文摘要 …… V

英文摘要 …… VI

目錄 …… VII

表目錄 …… IX

圖目錄 …… X

第一章　前言 …… 1

第二章　XML 的儲存結構與查詢 …… 8

　第一節　XML 的結構與儲存 …… 8

　第二節　規則路徑表示法 …… 13

　第三節　非決定性有限機制路徑索引 …… 14

　第四節　XML 的 API 應用 …… 18

　第五節　XML 的索引編碼與查詢 …… 22

第三章　雜湊函數編碼方法的 XML 查詢 …… 25

　第一節　雜湊函數的編碼 …… 25

　第二節　以雜湊函數為基礎的 XML 文件編碼 …… 28

　第三節　雜湊函數方法的查詢處理 …… 33

第四節　多元編碼與查詢 ……...……... 36
第四章　實驗結果分析 …….. 40
第一節　本論文方法與 s-DOM **方法的實驗比較** ……...……. 40
第二節　利用兩個雜湊函數的實驗分析 ……..…… 43
第三節　時間相關分析 ……...…… 46
第五章　結論與建議 ……...……... 50
參考文獻 ……. 51

表 目 錄

表 1 七個 XML 查詢語言的比較分析 …… 14

表 2 DOM 樹的個別節點編碼 *H* …… 30

表 3 DOM 樹向上編碼值 *S* …… 32

表 4 與 s-DOM 比較查詢 Shakespear 資料內容, 利用 PLAY.*[2].PERSONA …… 40

表 5 與 s-DOM 比較查詢 Shakespear 資料內容，利用*.TITLE 查詢句 …… 42

表 6 與 s-DOM 比較查詢 The book of Mormon 資料內容，利用 tstmt.*[1].(title|ptitle) …… 42

表 7 與 s-DOM 比較查詢 The book of Mormon 資料內容, 利用*.chapter …… 43

表 8 以兩個 hashing function 來查詢 Shakespear 資料內容，利用 PLAY.*[2].PERSONA 查詢 …… 44

表 9 以 兩 個 hashing function 來 查 詢 Shakespear 資料內容，利用*.TITLE 查詢 …… 44

表 10 以兩個 hashing function 來查詢 The book of Mormon 資料內容，利用 tstmt.*[1].(title|ptitle)查詢 …… 45

表 11 以兩個 hashing function 來查詢 The book of Mormon 資料內容，利用*.chapter 查詢 …… 45

表 12 時間實驗分析的四個查詢句與資料檔 …… 47

表 13 本論文方法與 s-DOM 方法比較效率提昇百分比表 …… 48

表 14 利用本論文方法的 1 個 hashing function 編碼和 2 個 hashing functio 編碼比較的效率提昇百分比表 …… 49

圖 目 錄

圖 1 XML 排板樣本範例 2

圖 2 XML 文件結構範例 3

圖 3 RPE 應用包含的範圍 5

圖 4 XML 文件及結構範例 9

(a) XML 文件範例 9

(b) DTD 文件定義範例 9

圖 5 RPE 查詢子句範例 13

圖 6 單一元素查詢的 NFA 圖 15

圖 7 父子相連查詢的 NFA 圖 15

圖 8 包含 or 條件式的 NFA 圖 16

圖 9 包含*號或?號的 NFA 圖 17

圖 10 利用 NFA 圖來表示 RPE 17

圖 11 SAX 應用的存取概念 19

圖 12 DOM 核心類別與介面模型圖 21

圖 13 對照圖 4(a)文件的 DOM 樹 22

圖 14 s-DOM 編碼範例 24

(a) s-DOM 個別節點編碼 24

(b) s-DOM 節點編碼值與進行 or 運算的編碼值 24

圖 15 Search Filter 圖例 26

圖 16 簡單 Bit Map 範例圖 27

圖 17 Bloom Filter 概念應用 27

圖 18 將 DOM 樹的各別節點加以編碼 29

圖 19 向上記錄圖 …… 31
圖 20 查詢步驟圖 …… 36
圖 21 多元編碼圖示 …… 38
圖 22 多元雜湊函數的編碼查詢範例圖 …… 39
圖 23 與 s-DOM 比較查詢 Shakespear 資料內容實驗結果，利用 PLAY.*[2].PERSONA 查詢句 …… 41
圖 24 與 s-DOM 比較查詢 Shakespear 資料內容實驗結果，利用*.TITLE 查詢句 …… 42
圖 25 與 s-DOM 比較查詢 The book of Mormon 資料內容，利用 tstmt.*[1].(title|ptitle) …… 42
圖 26 與 s-DOM 比較查詢 The book of Mormon 資料內容，利用*.chapter …… 43
圖 27 以兩個 hashing function 來查詢 Shakespear 資料內容，利用 PLAY.*[2].PERSONA 查詢 …… 44
圖 28 以兩個 hashing function 來查詢 Shakespear 資料內容，利用*.TITLE 查詢 …… 45
圖 29 以兩個 hashing function 來查詢 The book of Mormon 資料內容，利用 tstmt.*[1].(title|ptitle)查詢 …… 46
圖 30 以兩個 hashing function 來查詢 The book of Mormon 資料內容，利用*.chapter 查詢 …… 46
圖 31 編碼空間與 page I/O 數的實驗結果圖 …… 47
圖 32 編碼空間與查詢時間的實驗結果圖 …… 48

第一章　前言

隨著網際網路與電子商務的興起，傳統的資料處理方法也隨之而改變，演變成為可跨平台式的資料分享交換模式。XML (eXtensible Markup Language)的崛起，對於企業而言，更是一種強大的衝擊，因為 XML 有別於一般傳統的資料表示方法，更重要的是，XML 提供了一個跨平台的機制，不但可以用來交換不同平台的資料，還可用以管理資料與儲存資訊，解決了以往不同平台間，為了達到資料交換共享，所需支付的高昂成本。XML 是由一群 SGML (Standard Generalized Markup Language)專家與 WEB 專家，主要為了改善 HTML 不足與結合 SGML 優點而共同制定。在 1990 年以後，有許多 XML 相關的標準被提出，但自全球資訊網協會 (World Wide Web Consortium, W3C)於 1998 年公佈了 XML 1.0 的規格[34]開始，XML 已經成為國際間共同認定的資料表示交換標準，此標準也已被微軟、網景等知名公司所認同與採用。

XML 的應用層面也極為廣泛，不但制定一些標準的資訊分享的機制，也讓一些不同格式的資料，能夠被整合至同一個標準格式來分享利用。不但如此，還可以利用同一份的 XML 文件內容，套用在不同的排版樣本(Cascading Style Sheet)上，產生不同的展現樣式，展示在不同的網路平台介面上，如圖 1 所例。目前市面上對於 XML 技術方法的引用也非常的普及，像目前最流行的電子商務 EC、醫院之間的病歷資料分享、企業間不同作業平資料的交換轉換分享、PDA、行動電話 WAP 及一些在應用程式被儲存或交換的文件等等，如今 XML 已漸漸的成為網際網路間共通的語言與溝通的媒介。

XML 因為提供了跨平台的機制，因此被廣泛的使用。XML 之所以能夠跨平台式的資料交換，因為 XML 與傳統的連續性資料表示法有所不同，XML 著重在文件上的結構描述，舉個例子來說明，假設「 <學生>李建圖</學生> 」此句為一小段 XML 文件內容，其中<學生>是用來描述李建圖的資料內容，代表著李建圖是一個學生，而不是老師或其他。這樣的文件結構描述，再透過文件定義 DTD (Document Type Definition)來對標籤(tag)加以定義(<學生>)，就可讓參考同一開放式(public) DTD 的不同平台，透過小小的處理來交換彼此的資料，如此一來，每個不同平台對於<學生>tag 的定義就都相同了，而不會產生同樣字面而不同意義的情形。此類的標籤表示法與傳統的 HTML 相似，但卻存在著許多的不同點，如結構的嚴謹與分明、使用者可自訂其標籤內容及開使(<..>)與結束(</..>)tag 的成對使用等，這些特徵讓 XML 的運用更為廣泛，機乎所有的資料文件都可以利用 XML 的階層性結構來表示其內容，除了少部分的邏輯性結構以外。圖 2 為一個簡單的 XML 文件範例，由圖 2 就可觀查出 XML 文件的階層性結構與嚴謹程度。

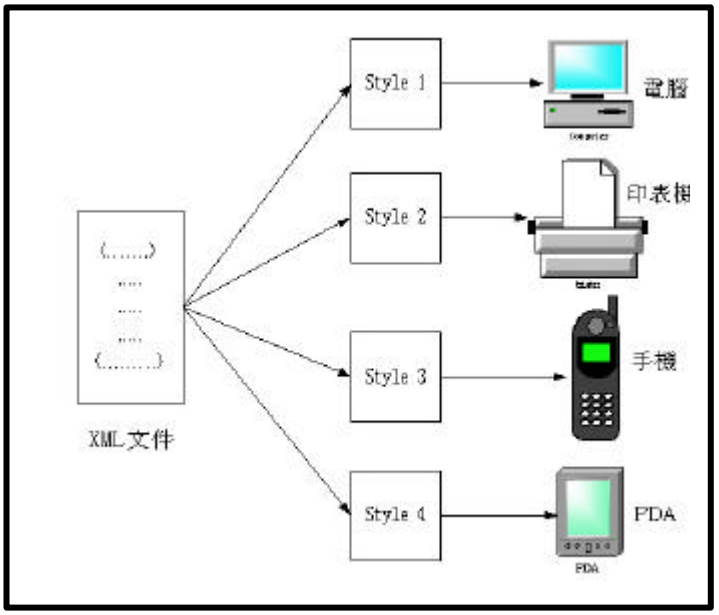


圖 1：XML 排版樣本範例

當使用了 XML 為儲存機制後，對於 XML 資料所使用的儲存媒介，就成了相當重要的問題，目前針對 XML 的儲存方式大約可分為三大類，第一類是儲存為 XML 的檔案形式，此類較少人使用。第二類是將 XML 儲存於關聯式的資料庫中，也是目前企業界最為廣泛採用的方法，現今市面上的資料庫軟體廠商機乎都有提出自已的儲存機制，且紛紛投入大量資源來從事 XML 的研究，如 Oracle、IBM DB2、MS SQL Server 等。第三類是儲存於半結構化(Semi-structure)的資料庫或是原生性資料庫(Native XML Database, NXD)之中。NXD 是近幾年才崛起的資料庫，如 Software AG 公司發行的 Tamino 系統等。NXD 資料庫的崛起解決了利用關聯式資料庫儲存 XML 資料的不完全融合問題，因為 NXD 資料庫不像關聯式資料庫般，需要將 XML 的階層性資料加以分解，分解成可儲存於關聯式資料庫的資料表之中，NXD 能以物件的方式，以整分文件為單位來儲存 XML 的資料，不需轉換，不需分解，直接儲存整個 XML 文件的資料與定義。NXD 的崛起，也讓 XML 的研究更往前邁進了一大步，就目前 XML 的儲存研究領域問題中，如何在關聯式資料庫與 NXD 資料庫之中，做彼此資料的直接轉換應用，而且是毫無轉換誤差的情況，是目前廣泛存在的問題，也是所有使用關聯式資料庫來做為儲存平台，轉換

```
<University>
    <School> School 1
        <address>…….</address>
        <tel>…….</tel>
        <dep.>……..</dep.>
        <dep.>……..</dep.>
    </School>
    <School>
        ……
        ……
    </School> School 2
    ……
    ……
</University>
```

圖 2：XML 文件結構範例

到以 NXD 為儲存平台時，必經的一個必要過程。

雖然 XML 提供了一個共同的資料格式交換標準，但在資料量的累積日益擴大之下，對於 XML 而言，所要面臨的不再只是儲存或轉換的問題而已，衍生出來的便是如何在這些龐大的 XML 資料中，迅速有效的找出使用者所需要的資訊，不論 XML 資料是儲存於關聯式資料庫或 NXD 資料庫中，查詢的效率，儼然已成為一項重要的研究問題。

對於 XML 的查詢技術，至目前為止仍是一項炙手可熱的研究重點，XML 的查詢技術約略可分為兩大研究方向，第一類是針對儲存於關聯式資料庫中的 XML 資料，做查詢的方法技術，第二類是半結構化或物件式資料庫的查詢處理技術，如 NXD 資料庫。在第一類的方法裡，所要處理的是關聯式資料庫以資料表來儲存 XML 的方式，資料表的儲存結構原本就與 XML 的階層性資料結構有所差異，在儲存時通常會將 XML 的結構定義與資料分解各自儲存，再利用指標或關聯的方式，來判斷 XML 的資料階層關係或來呈現資料內容。在查詢上也必須透過一些反覆轉換 join 的機制，來完成階層性資料的查詢對應(mapping)[10, 11, 16, 26]。第二類的方法就符合 XML 層階性的資料表示法，雖然 XML 並非物件式的資料，但卻可以利用物件的概念來處理 XML 階層性資料內容，像 NXD 就是為了解決儲存 XML 資料所發展出的資料儲存結構[20, 29]，當查詢的技術能夠運用在第二類的方法時，不論在於技術的引用與效率的展現，成果都更為驚人。

在這些 XML 的查詢研究領域中，大部分的查詢技術都支援規則路徑表示法(Regular Path Express, RPE)，且支援 RPE 的查詢技術或查詢語言，是眾所認同較有價值的 XML 查詢方式。規則路徑表示法 RPE 是一種 XML 的資料查詢語法規則，RPE 是利用一些簡單易懂的文字或符號，來

表達 XML 的階層性資料結構[1, 6, 8, 23]，像是「?」、「.」、「*」 等符號，再與 like-SQL 的查詢句結合運用。這些符號各自代表著不同的意義，可用於一些無法得知資料全貌時的查詢，利用一些符號的描述來代替階層之間的關係，如「school.*.litoto」的查詢句，「*」號就代表 school 與 litoto 中間的任何層次與關係。支援 RPE 的 XML 查詢語言有很多，如 Lorel[1]、Xpath[35]、XQuery[36]、XQL 等，當然這些 XML 查詢語言的設計都是符合 W3C 所提出的標準規範，也各有不同的特色與支援的程度範圍。

RPE 的應用極為廣泛，不論在 XML 應用系統的建制或查詢語言的技術方法，極少是沒有支援 RPE 的運用。在 XML 的研究領域裡，機乎所有研究學者的研究方向，也都尊循著 W3C 所提出的標準來進行研究。圖 3 為 RPE 的運用層面圖，最內層是 XML 的儲存機制，中間層是支援 XML 儲存機制的 XML 查詢語言，最外層是 RPE 層，在 RPE 層內的所有層級，都包含於 RPE 的規範標準中。目前針對 XML 的查詢語言研究，已算是進入非常成熟的階段，近幾年所提出的一些 XML 查詢語言，都是遵循 RPE 的規範來設計，這些所被研究提出的 XML 查詢語言，也都被廣泛的利用在一些 XML 應用程式或查詢索引中，如 eXcelon Information Server、

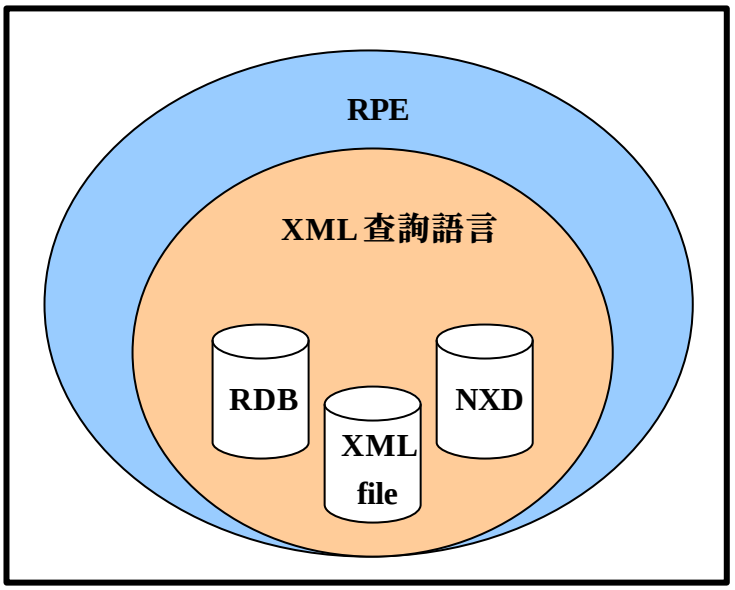


圖 3：RPE 應用包含的範圍

Oracle 等系統的應用。

在 XML 的查詢語言與儲存機制的研究日趨成熟下，查詢時的效率就愈顯的重要及被重視，這些年有許多學者也運用了一些 XML 的查詢語言，提出一些 XML 查詢檢索技術，例如運用一些索引(indexing)、編碼(encoding)及指標鏈結等方法，如典型的 T-index[23]、Lore[20]的索引技術方法等。這些被提出的查詢檢索方法，其最主要的目地，都是希望能夠快速且直接有效的改善檢索時的效率，降低查詢所花費的時間或花費的成本，不論這些檢索的方法是運用在關聯式的資料庫或是 NXD 中，效率都是大家所重視的問題。但在這些眾多的查詢檢索技術中，往往都有其缺失或不足的地方，典型的問是如查詢結果無法包含全部所需要的資料內容，或是無法全範圍式的搜尋，這些缺失往往導致查詢結果會有遺漏重要資訊的遺憾。

在這些被提出的檢索技術中，大部分所運用的概念都是以 pattern tree match [9, 13, 16]的方式，來比對判斷查詢句的路徑與 XML 資料的階層結構是否吻合。另一方法是 XML API 的運用，此類查詢方式比較？向於樹的拜訪方式[23, 25]，但卻只能運用在物件式或半結構化的資料儲存方法上，而支援 XML API 的程式語言有很多，如 Java、C++、perl 等。目前的 XML 研究中，較多人採用的 API 項目有兩種，第一種為文件物件模式(Document Object Model, DOM)的方式，第二個為 XML 簡單的 API (Simple API for XML, SAX)應用的方法，這兩個方法雖然都提供了同樣的目的，但在功能與作業處理的方法上，各有著些許的差異性與特色。

在本論文中，提出一項查詢檢索的技術，利用雜湊函數(hashing function)的編碼技術，但必須應用在資料異動性較小的 XML 資料查詢上，同時也支援 RPE 的查詢方式。本論文的方法運用了 XML API 的 DOM

樹模型，透過 hashing function 的技巧，來處理 XML 文件的編碼與查詢時資料的核對，在對 XML 資料做查詢的過程中，能有效的裁減掉不必要的查詢處理過程，來提昇 XML 跨文件資料的全範圍查詢效率，是個極有效率的 XML 查詢檢索演算法。在本論文中提出的演算法，與[25]所提出的 s-DOM 查詢方法相比較，在 s-DOM 的方法上有一些缺點存在，當 s-DOM 的編碼結果為較差狀況時(如 11101110)，而較接近根節點的節點，與子樹整合後的編碼(指與節點的子樹進行 or 遲算)值，可能導至大部分都為 1 或全部皆為 1(如 11111111)，因此在執行查詢比對運算時，容易造成全部都成立的狀況，詳細內容請參考第二章第四節。本文提出的方法，不但改善了 s-DOM 方法在編碼上的缺點與不足，更能有效的減少在查詢時所產生錯誤判斷比率。實驗結果顯示出，本論文所提出的方法，查詢效率高過於 s-DOM 的查詢方法，效率約提昇了 51%。

本論文的其餘架構如下：第二章探討一些 XML 相關的技術背景，第三章介紹如何將規則路徑表示法轉換為簡單的路徑索引，稱之為非決定性有限機制 NFA，第四章介紹應用在本論文提出的演算法，本論文的方法是利用 hashing function 技巧來作索引及查詢，第五章提出本論文方法的實驗結果比較與分析，第六章為結論探討。

第二章　XML 的儲存結構與查詢

本章節將會深入討探本論文所提出方法的背景技術與理論探討。第一節先簡介 XML 的結構及目前儲存 XML 資料的機制，第二節簡介關於 RPE 的運用及一些 XML 查詢語言的比較，第三節深入介紹目前被廣泛使用的 XML API 模型，第四節我們將介紹與本論文所提出之查詢方法比較的另一查詢方法，稱為 s-DOM[24, 25]。

第一節　XML 的結構與儲存

XML 與 HTML 皆是一種標示語言(Markup Language)，所謂的標示語言，是指由一些標籤(tag)或類似標籤的編碼(codes)所組成的表示方法。而標示語言並不能獨立的來使用，它必須是與一些有意義內容的文件資料結合，才會變的有意義，且當文件使用標示語言來表示時，文章資料內容會變成有階層性結構的資料表示法，如圖 4(a)。一般而言，標示語言可分為兩類，第一類是因特別用途而設計的標示語言，如 HTML 與 XML 就是此類型的標示語言，皆是為了 Web 上的應用而設計。第二類是通用型的標示語言，如 SGML[2]，這類型的標示語言，通常可以讓不同的軟體來解讀，而不是為了某一特定軟體而設計。

就目前在 XML 的儲存方法上分為三大類，最原始的方法是將之儲存為檔案的格式，後來有人運用以物件式的資料庫來儲存 XML 資料，但目前較為廣泛流行的趨勢是利用企業內既有的關聯式資料庫，透過將 XML 的原始資料，轉換至可儲存於關聯式資料庫的資料表中，或將儲存於關聯式資料庫中的 XML 資料，轉換成原始的 XML 文件的方法，使 XML

資料可於關聯式的資料庫中進行處理。但近幾年又崛起新的資料庫型態稱之原生性資料庫(Native XML Database, NXD)，這類型的資料庫是不需透過轉換的過程，可直接儲存 XML 樹狀階層式的資料型態內容。以下分別來說明探討 XML 的結構表示法與 XML 的資料儲存機制。

壹、XML 文件結構

XML 與 HTML 相較之下有些雷同，但仍與 HTML 有許多的差

```
1  <?XML version=”1.0” standalone=”no”?>
2  <!DOCTYPE PHONE PUBLIC “ex.dtd”>
3
4   <PERSONAL>
5       <PERSON>
6            <NAME> Litoto </NAME>
7             <DEGREE> A </DEGREE>
8             <MAIL> g0112010@mail1.nhu.edu.tw </E-MAIL>
9             <TEL> 052721001 </TEL>
10      </PERSON>
11      <PERSON>
12           <NAME> DEF </NAME>
13           <DEGREE> B </DEGREE>
14           <MAIL> litoto@mail1.nhu.edu.tw </E-MAIL>
15           <TEL> 052721002 </TEL>
16      </PERSON>
17 </PHONEAL>
```

圖 4(a)：XML 文件範例

```
1  <!ELEMENT PERSONAL ((PERSON)+>
2  <!ELEMENT PERSON (NAME, DEGREE, MAIL, TEL*, CELL*)>
3  <!ELEMENT NAME (#PCDATA)>
4  <!ELEMENT DEGREE (#PCDATA)>
5   <!ELEMENT MAIL(#PCDATA)>
6  <!ELEMENT TEL (#PCDATA)>
7  <!ELEMENT CELL (#PCDATA)>
8   <!ELEMENT DEF  “C-T Li”>
```

圖 4(b)：DTD 文件定義範例 ex.dtd

異性存在，如 XML 文件結構上的規範嚴謹度(成對的 tag 使用)、對文件的自我描述能力(自行定義 tag 內容)、文件定義 DTD 的引用[14, 34]等。XML 之所以在近幾年來被廣泛的使用及研究，是因為他有著與傳統標示語言 (Markup Language)不一樣的特性與能力。尤其是 XML 的結構規範與文件格式定義(DTD)的運用，可以讓使用者能夠順利的在不同平台上，交換彼此所需要的資訊內容，達到跨平台的資料交換特性。圖 4(b)為一個簡單的 XML DTD 檔範例；圖 4(a)為引用圖 4(b) DTD 檔的 XML 文件範例。

由圖 4我們就可以觀查出 XML 文件的結構嚴謹性，而且標籤(tag)也可以由使用者自行訂義，來描述文章的內容資訊，圖 4(a)的 XML 文件檔案引用了圖 4(b)的 DTD 格式(XML 的文件定義也可以直接包含於 XML 文件檔案之中或省略文件的定義，但較少人採用)，圖 4(a)的第 2 行就是引用圖 4(b)文件定義檔的宣告。使用者可以利用 XML 的文件定義來對所有的 XML 文件的元素或屬性內容加以定義與規範，就如同程式語言對於變數宣告的概念般，只要此文件定義為開放式(public)，就能讓不同平台間的作業處理，透過網路來引用此共同的 DTD 內容，而只要是遵循同樣 DTD 的所有不同平台處理端，對於同樣 tag 內容的資料訂義就都相同了，既使所有 XML 文件的格式並不相同，也能輕易的達到跨平台資料交換的目地了。

我們可以利用 DTD 的訂義，來針對每個元素名稱，定義其元素或屬性的型態(如圖 4(b)的括號( )符號內容)、可允許的出現次數(如圖 4(b)第 2 行之*符號或?等)、結構(如圖 4(b)第 2 行)及預設值(如圖 4(a)第 8 行及圖 4(b)的第 12 行)等。這些功能都是 HTML 所無法達成的功能，雖然 XML 與 HTML 都是 SGML (Standard Generalized Markup

Language)上應用的延申[37]，但功能卻更強大，使用也更為簡單，應用更為廣泛，這些功能能夠輕易的取代傳統的 Cold Fusion 或 ASP 的自訂標籤技術，這就是為什麼 XML 會愈來愈普及被採用與研究的原因。

貳、XML 的儲存

目前針對 XML 儲存的機制，最常見的有三種：第一種為檔案型態，第二種為儲存於關聯式資料庫中的資料表方式，第三種是儲存成在半結構化物件式或原生性資料庫(NXD)的型態，我們分別來探討這三種儲存的機制與方法。

(一)、XML 檔案(XML file)

XML 檔案雖然附檔名為點 XML，但實際上就是普通的文字檔內容, 卻可以利用瀏覽器來剖析(parser)查看。雖然將 XML 儲存為一般的檔案比儲存在資料庫容易與方便，也絕對支援 RPE 的方法，但卻也存在許多缺點。典型的像是資料量過於龐大的問題、對於檔案內容索引或及時交易處理等等問題。所以一般而言，對於 XML 的儲存，極少人會以檔案為儲存的機制，即使有，絕大部分都只是用於暫存。

(二)、關聯式資料庫系統(Relation Database System)

關聯式資料庫是目前市面上運用在儲存 XML 資料最廣泛的方法，如 Oracle、IBM DB2、Informix、Sybase 等資料庫。有許多學者都紛紛針對 XML 如何儲存於關聯式資料庫，深入研究探討其可用運用的方式與轉換技術。但對於 XML 的階層性樹狀資料結構而言，要將 XML 資料內容儲存於關聯式資料庫的資料表中，必定存在一定的困難度與不相容。

XML 的階層性資料表示結構與傳統關聯式的資料表儲存方法是有所差異，若要將 XML 資料儲存於資料表中，則必須經過一套轉換的機制與標準[5]，才能將 XML 資料內容或含有文件定義 DTD 的 XML 文件，儲存於關聯式資料庫的資料表之中。但在轉換原始 XML 文件至關聯式資料表過程中，或將儲存於關聯式資料表的 XML 資料，經過反覆處理轉換的動作(如 join、link 等)，將之轉換還原為原始的 XML 資料內容時，會造成與原始資料有誤差或不一致的情況產生，而要利用資料表來儲存階層性的資料內容，就必需要將階層資料都分解成獨立的狀態，再透過一些技巧，來辨別在資料表中的每筆資料彼此間的關係。這仍是個值得研究的方向，因為在大部分的企業中，仍然保有關聯式的資料庫系統為主，目前已經有許多學者提出相關儲存規則與技術方法[10, 11, 16, 26]來決解這類的問題。

(三)、原生性資料庫(Native XML Database, NXD)

NXD 為最近幾年提出的一種針對 XML 資料儲存的機制，目前市面上已經有少部分資料庫軟體公司，研發出相關的 NXD 資料庫系統，如 eXcelon 公司的 eXcelon Information Server、Software AG 公司的 Tamino 與 Oracle 公司所推出的 XML DB。

NXD 與關聯性資料庫，在儲存 XML 資料上有著極大的差別存在，原生性資庫對於 XML 文件的儲存是以一份 XML 文件為儲存單位，直接儲存 XML 樹狀階層性的資料內容與 XML 文件的 DTD。相較於關聯式資料庫而言，NXD 不用將 XML

文件做分解，存取也不用經過特殊的處理，的確是更適合用來儲存 XML 文件資料，在資料的處理上，也更容易與技術結合且更方便，不至於會產生與原始資料不符合或不相容的情形。

在 XML 儲存方法的研究領域中，由於 NXD 的崛起到目前為止的時間並非很長，所以在 NXD 的儲存效率及方法的研究比較[1, 7]，並非有太多的成果提出。但仍然有部分針對 XML 的檢索詢查與系統建構的研究，是 NXD 為後端儲存的架構，相信仍然有更多的空間是值得再深入研究探討。

第二節　規則路徑表示法

所謂規則路徑表示法(Regular Path Expression, RPE) [1, 6, 8]，是能夠以 like-SQL 語言的方法表示查詢的內容，但卻可以結合一些特殊字元或一些特殊符號，來描述 XML 文件的階層式及樹狀資料內容，如樹狀結構的子孫關係的描述等。在利用 RPE 來下達一些 like-SQL 的指令時，不一定要完全知道目前所查詢的 XML 文件資料全貌長像，只要針對所要查詢的資料條件，加以利用 RPE 的一些符號或特殊字元來與條件結合描述，就可以輕易的查詢出所要尋找的 XML 資料內容或一些資料關係。圖 5 就是一個簡單的 RPE 的查詢範例。

```
PHONE.(PERSON.*).DEGREE
```

圖 5：RPE 查詢子句範例

圖 5 的規則路徑表示法查詢子句(查詢內容參考至圖 4(b)之 XML 文件內容)，主要目的是找出 PHONE 的下一代為 PERSON，且 PERSON 的任一子孫中(*號)有出現 DEGREE 的元素(element)。由圖 5 我們可以觀查

出，在 RPE 中，使用者未必要全然知道目前所有查詢 XML 資料的長像，也能透過類似圖 5 的規則路徑關係描述方法，加上一些簡單的符號來表示，就可以簡單的找出使用者所需的資料內容與關係。

支援規則路徑表示法的 XML 查詢語言有很多，當然也有不支援規則路徑表示法的 XML 查詢語法，但通常是較早期提出或針對特殊目地而設計。在規則路徑表示法上，各查詢語言所支援的程度上也有所差異[3]，我們以表 1 來做分析比較。

XML 查詢語言 / 支援使用方法	LOREL	XML-QL	XML-GL	XSL	XPath	XQuery	XQL
利用萬用字元來表示未知的結構路徑	是	是	部分可以	是	是	是	是
查詢終止條件	是	未定義	是	否	是	是	未定義
支援建立修改內容	是	是	是	是	是	是	否
巢狀式查詢	是	是	否	是	是	是	否

表 1：七個 XML 查詢語言的比較分析

第三節 非決定性有限機制路徑索引

非決定性有限機制(non-deterministic finite automata, NFA)是一種可針對規則路徑表示法 RPE，做查詢路徑估計與計算的法則[17]。任何復雜的規則路徑表示法，都可以將其轉換為相對應的 NFA 來表示。換句話說，也就是將其轉換為相對應的路徑索引，並運用在規則路徑表示法查詢上。本節將介紹四種組成 NFA 的表示法方及如何將 RPE 轉換成對應的 NFA 表示。

壹、四種組成 NFA 的表示方法

對於 NFA 在規則路徑表示法上的運用，其時間復雜度為 $O(|S|^2|RE|)$[29] (*S* 為 NFA 的狀態數(state set)，*RE* 為規則路徑表示法 RPE 的長度總數)。所有的規則路徑表示法，都可以由下列 4 個 NFA 表示方法來組成。

L (one-set)：

當規則路徑表示法為查詢單一元素時，如查詢 student 一個單純元素或屬性時，此時可以利用圖 6 的表示方法，來表達這類型的查詢[17]，圖 6 中之 r 便是 student 內容。

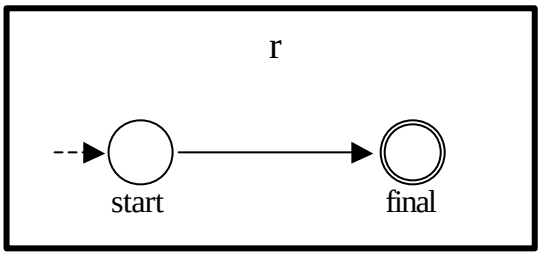


圖 6：單一元素查詢的 NFA 圖

L (r_1) L (r_2)：

規則路徑表示法查詢句中，若有父子相連的查詢子句內容產生，則可利用圖 7 的 NFA 圖來轉換表示這類的查詢[17]，對照如圖 7，我們可以將之視為 $r_1.r_2$，把 r_1 視為父代，r_2 視為子代的相連查詢 NFA 表示方法，而 *e* 是指經過一個 edge，也就是指從父節點穿越一個 edge 到達子節點。

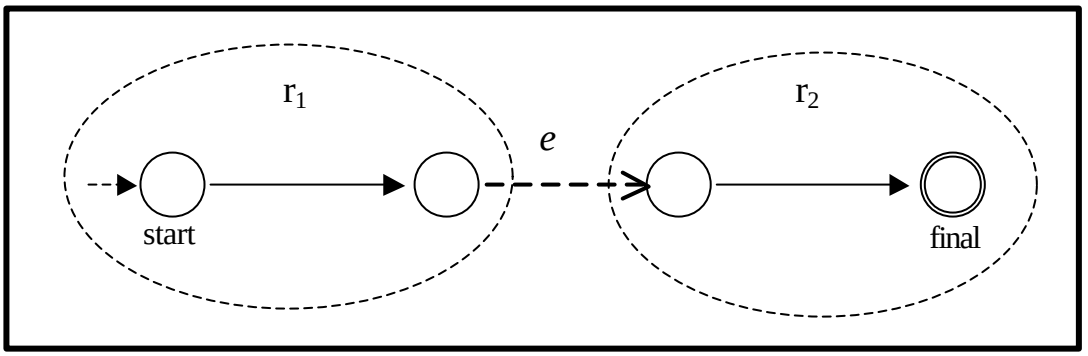


圖 7：父子相連查詢的 NFA 圖

L (r_1+r_2)：

當規則路徑查詢包含 or **查詢條件內容時，可將其轉換為圖** 8 **對應的** NFA **表示法**[17]**。如** (r_1 | r_2)**，其中「**r_1 | r_2**」便是一個** or **查詢條件的範例，也就是說只要有出現** r_1 **或** r_2 **的元性或屬性，都符合條件式。**

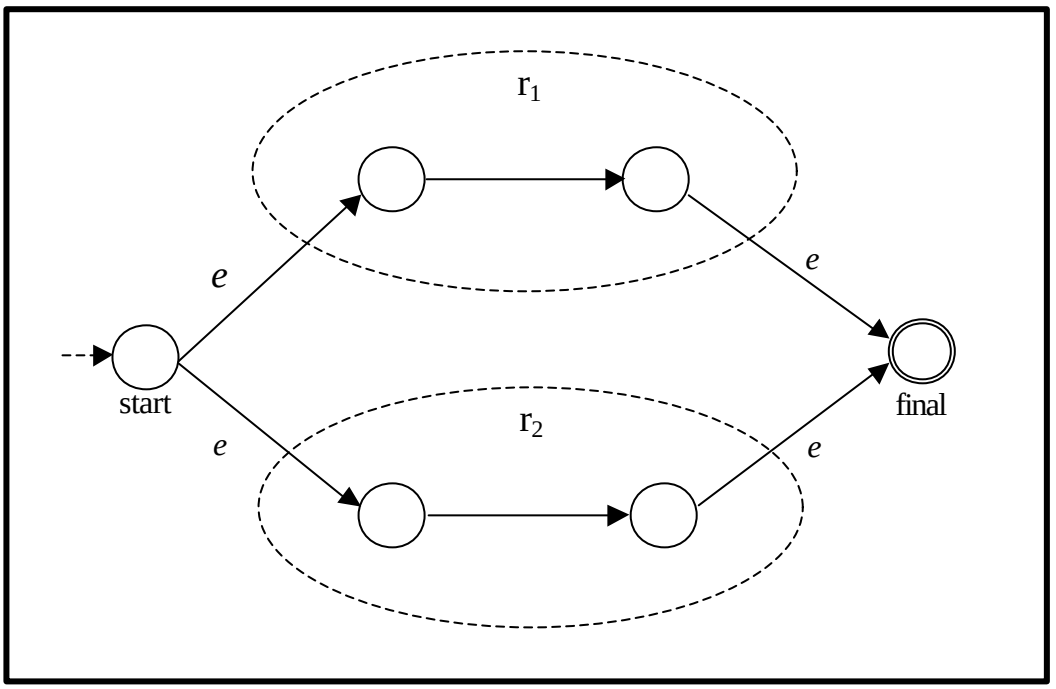


圖 8**：包含** or **條件式的** NFA **圖**

L (r*)**或**(r?)：

規則路徑查詢允許一些較模糊的查詢符號存在，但這些查詢符號卻也代表著一些查詢的重要條件因素，不可乎視。圖 9 **便是一個模糊規則路徑查詢的對應** NFA **表示圖**[17]**，舉個例子來說，如** r_1 .*. r_2 **的查詢句，主要目地是要找出** r_1 **節點下，不論經過幾代，只要曾經出現過** r_2 **的元素或屬性，皆符合查詢的條件。其中「*」符號即是個模糊的查詢符號，類似此查詢句我們都可以利用圖** 9 **來轉換成** NFA **的表示。**

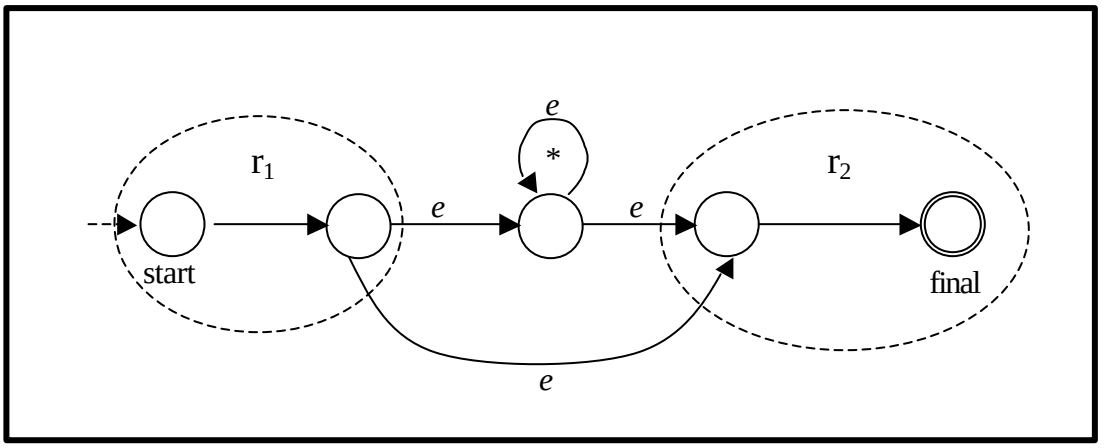


圖 9：包含*號或?號的 NFA 圖

貳、 RPE 與 NFA 的轉換

上一小節是介紹有關 NFA (non-deterministic finite automata)轉換的 4 種機制，不論多複雜的 RPE 查詢，都可以利用 NFA 的 4 個圖來表示，將之轉換成以 NFA 來表示的查詢索引圖。圖 10(a)為一個 RPE 的範例，主要目地是查詢 DEGREE 或 NAME 元素，並且必

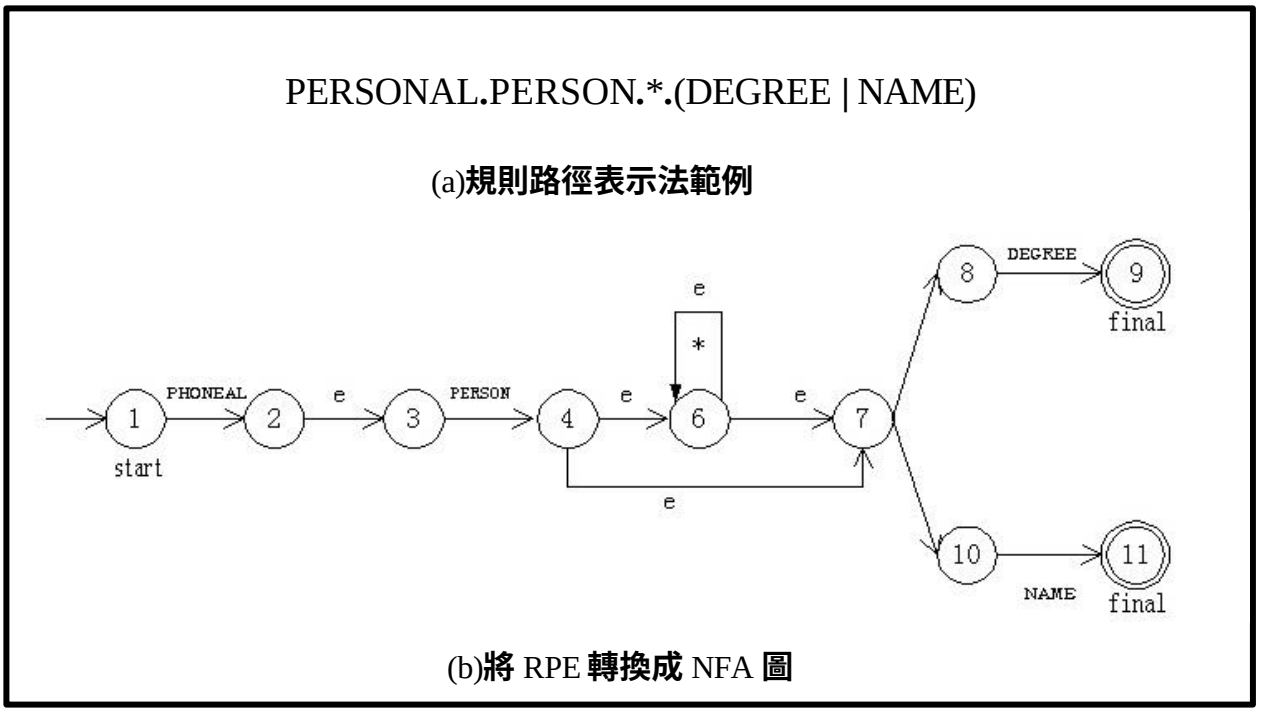


圖 10：利用 NFA 圖來表示 RPE

須是存在於 PERSONAL 是 PERSON 的父親的條件下的所有 DEGREE 或 NAME 內容。圖 10(b)是將圖 10(a)的 RPE 查詢句，轉換成以 NFA 圖來表示。

圖 10(a)的 RPE 句，經由上一小節所介紹的 4 個 NFA 轉換法則轉換後，成為圖 10(b)的 NFA 表示圖，來表示 RPE 的索引。舉個例子，假設圖 10(b)開始的狀態為 SS={1}，result set 為空，接著先擷取 DOM 樹編號為 1 的根節點，此時 label path 為{PERSONAL}，SS={3}。然後 DOM 樹編號為 1.1 被擷取，此時 label path 為{PERSONAL.PERSON}，SS={6, 8, 10}。接著 DOM 樹編號為 1.1.1 的節點被擷取，此時 label path 為{PERSONAL.PERSON.NAME}，SS={6, 10, 11}，而 NFA 編號為 11 為 final 節點，所以代著 DOM 樹節點編號為 1.1.1 節點被接被，result set={1.1.1}。這此方法類推，既可將所有作業完成。當利用此轉換後的 NFA 來進行規則路徑的查詢作業，能很容易的在 XML 文件的 DOM 結構樹上做簡單的路徑估計與索引[15, 17, 29]，提昇了傳統規則路徑的查詢效率，也避免了一些不必要的時間浪費。

第四節　XML 的 API 運用

XML 的 API (Application Programming Interface)應用，大約可分為幾類，SAX (Simple API for XML)與 DOM (Document Object Model)等。

壹、XML 的簡單 API (Simple API for XML, SAX)

XML 簡單 API 應用(SAX)為 XML 的 API 應用方法之一[28]，其主要以事件為基礎(Event-based) [21]，與 DOM 有著極大的差異性，但使卻較簡單，也有較多的缺點存在。

當我們使用 SAX 來處理 XML 階層性資料時，我們能把 XML 文件視為一棵樹，而 SAX 不需要去知道整棵 XML 樹的蓋觀，如圖 11 所表示。因此在運用 SAX 必然會出現較多的不方便性，功能也較不齊全。因為 SAX 為序列式模型(sequential model)方法，所以無法隨機去取得 XML 樹的任一節點，必需要觸發 SAX 樹的某部份節點事件後，才能夠取得 XML 樹所觸發的部份節點內容，且只能以循序的方式來處理。雖然有著上述的不方便性，但卻也有其優點存在，如佔用較小的記憶體空間，適合運用於較淺或較小的 XML 樹，在較小的 XML 樹上處理效率會高過於 DOM 的方法。

雖然 SAX 的應用是以事件為基礎，並非以全樹為基礎，但我們仍然可以利用樹的型態來探討 SAX。由圖 11 我們可以看出來，當我們利用 SAX 來處理 XML 樹時，因為是以事件為基礎，所以不需要全盤了解整棵樹的形狀，假設目前狀態在節點 1.2，SAX 的方法只要知道 1.2 的下一代既可。換句話說，只需要把接近於節點 1.2 的其它節點放進記憶體中。由於這個因素，所以利用 SAX 的方法會較節省記憶體，但卻無法隨機取得其它節點。

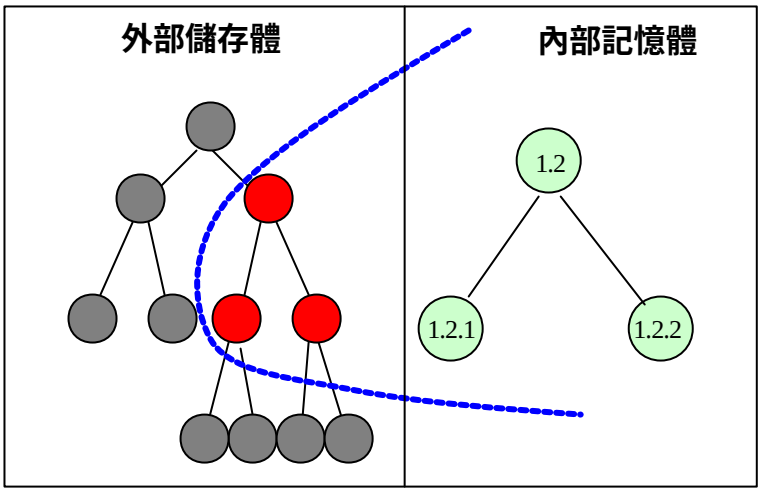


圖 11：SAX 應用的存取概念

貳、文件物件模型 (Document Object Model, DOM)

文件物件模型(DOM)為 XML 另一 API 的應用[27]方法，與 SAX 有許多的差異點，SAX 以事件為基礎(Event-based)，但 DOM 卻是以結構樹為基礎(tree-based) [21]，換句話說，SAX 是將所需要到的事件放入記憶體中，但 DOM 卻是將整棵樹放入記憶體。由此可以想像，若此樹極為龐大或者說此樹漸漸的在擴大，那麼對於處理上是一項較大的負擔，且 DOM 的方法比起 SAX 方法，佔了較多的資源。

相對而言，DOM 也有許多優點是 SAX 無法媲美，對於 DOM 而言，所擁有的功能與所能處理的事件，遠遠超越 SAX。DOM 不但能跨文件的處理，且由於是將整棵樹放入記憶體中，所以能自由隨機的存取樹上任一節點。

DOM 是 W3C 所認同的標準之一，不但能用來處理 XML，也能處理 HTML 或其它相關的標示語言，圖 12 為 XML DOM 的一些核心類別與可運用的介面模型[33]。目前 DOM 被組織成 3 個層次，第一層級(DOM Level 1)是定義了對樹操控的功能及一些內部機制[32]，且 DOM Level 1 也已受 W3C 所認可。第二層級(DOM Level 2)是以 Level 1 為基礎，建構特別內容模型，且也改善 Level 1 之缺乏的功能定義。第三層級(DOM Level 3)仍在建構中，未正式推出。

支援 DOM 與 SAX 方法的程式語言有很多，最廣泛被使用的為 jave 與 C/C++，Oracle 8i 版本以後之 PL/SQL 也支援 DOM 與 SAX 的運用設計。在許多支援 XML 處理的軟體中，後端的程式設計也機乎都脫離不了 DOM 與 SAX 技術的採用，XML API 已是所有研究都共同認定的標準之一。所有的 XML 階層性結構的文件資料，都可以對應成為一個 DOM 的樹狀結構圖，圖 13 為圖 4(a)的 XML 文件對

應的 DOM 樹狀結構圖。

圖 13 是一個簡單標準的 DOM 樹狀圖，在這個圖中，我們可以把樹上每個節點視為一元素(element)、屬性(attribute)或內容(text)。元素(element)、屬性(attribute)與內容(text)是 DOM 類別較常見的類別，當然在 DOM 的運用不只只有這三種類別，還有更多核心功能的類別可加以運用，如圖 12 類別模組所示。這些功能模組的類別，各有其不同的功能與運用時機。

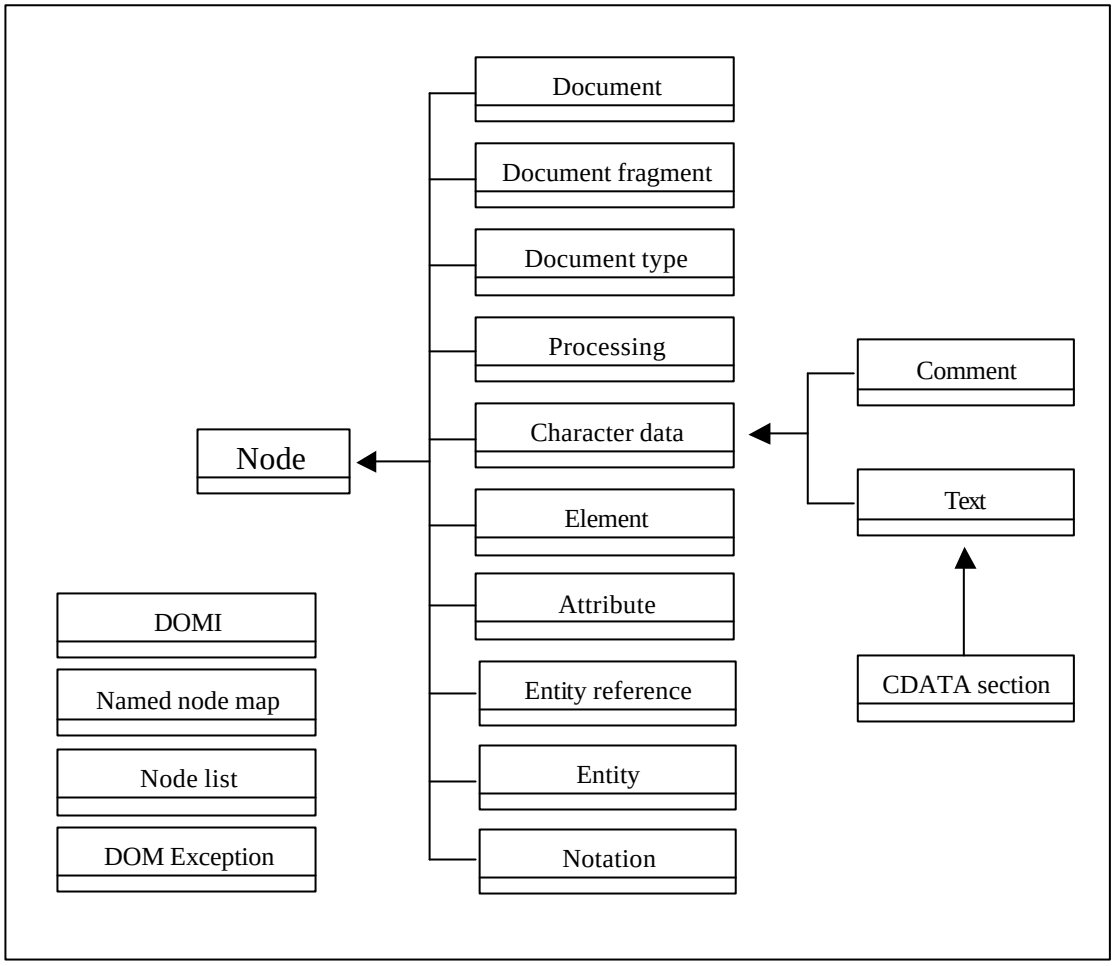


圖 12：DOM 核心類別與介面模型圖[33]

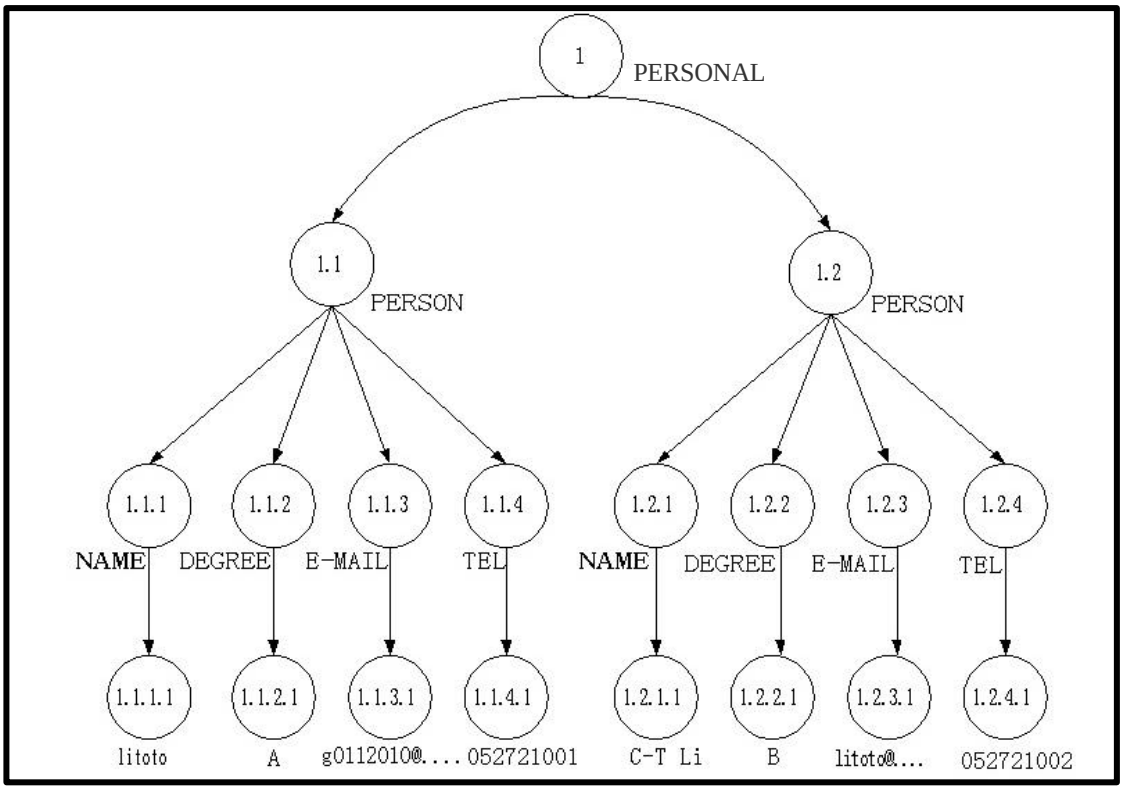


圖 13：對照圖 4(a)文件的 DOM 樹

第五節 XML 的索引編碼與查詢

近幾年來針對 XML **查詢處理上的研究，有著相當大的研究成果，在** XML **的索引與查詢部分有** Lore、T-index、EBIM[30]、s-DOM[24]、DataGuide[15]**等較知名的方法被提出，在這些查詢處理的方法中，各有其可以運用的優點，也各有其弊處，且與傳統資料庫系統索引機制有所差異。下一段將介紹與本論文所提出方法做實驗比較的另一查詢方法，此方法稱為** s-DOM**，並分析其方法的缺點。**

s-DOM [24, 25]**為** S. Park **及** H. Kim **於** 2001 **年提出之方法，此方法是將** DOM **樹的節點加以編碼，並運用於查詢的過程，做為查詢過濾的條件。透過此方法，能在對** XML **文件樹做查詢拜訪的過程中，先經過編碼的比對，來判斷是否要放棄目前節點下的子樹的拜訪，由此來裁剪**

不必要的過程。

假設有一 XML 文件的 DOM 結構樹如圖 14，圖 14(a)為一個個別節點的編碼範例圖，節點編號為 1.1 的 DOM 元素(element)節點，此元素節點的值為「student」，經由 s-DOM 的 8 bit 編碼大小，將編號為 1.1 的 DOM 元素節點內容值 student 加以編碼，經碼後的編碼值為「11010011」，我們由此編碼值可觀查出，其編碼有如 ASCII 碼的編碼方式。在 s-DOM 的編碼處理過程中，必須將節點的子孫節點，做 or 的運算。以圖 14(b)為例，假設節點編號為 1 的節點，就必須與節點 1.1、1.2、1.1.1、1.1.2. 及 1.1.1.1 的所有節點做 or 的運算，圖 14(a)每各節點中括號([])內的編碼值就是做 or 運算後的結果值。假設圖 14(a)為一要查詢的條件編碼值，但因編號節點 1 的 or 運算後值為 11111111，則 11111111 與查詢條件的編碼值 11010011 進行 and 運算，結果仍然是自已本身，也就是仍為 11010011，所以必需再進行子樹的拜訪，但事實並不存在其子樹中，因此容易造成錯誤的判斷及不必要的時間浪費。在 s-DOM 的編碼方法中，若在其子節點中，有一節點的編碼值是屬於較差的情況(如 11101111)，在進行 or 運算時，可能就會造成 8 個 bit 皆被填滿的問題產生(如 11111111)，節點編號為 1、1.1、1.1.1 的 DOM 節點，當這些節點與其所有子節點進行 or 運算後，結果都會是 11111111。這類的節果，會導至在做查詢運算時，所有的運算結果皆是成立。尤其當 XML 的 DOM 樹屬於較深的情況或分支較多時，這類型的問題更容易發生，查詢效率也會因此下降，既使加大其編碼的空間大小，編碼上仍然存在較差的編碼結果，並無法真正去改善此問題的發生。

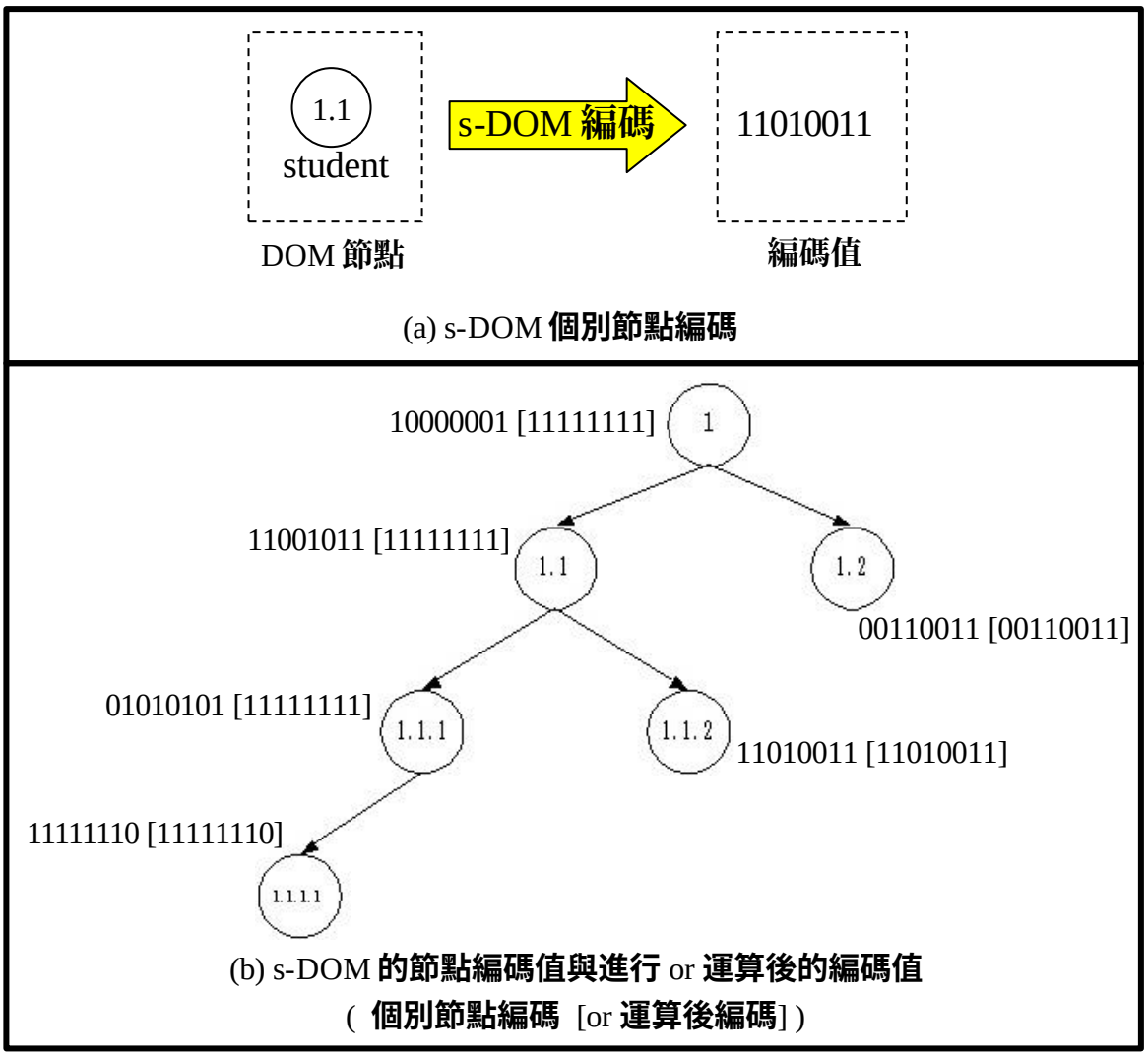


圖 14：s-DOM 編碼範例

第三章　雜湊函數編碼方法的 XML 查詢

本論文所提出的 XML 查詢方法，是以雜湊函數的編碼概念技巧，且必須應用於異動性較小的 XML 資料上。如第三章及本章上一節章所介紹，在本論文的方法裡，將規則路徑表示法的查詢(RPE)透過 NFA 的轉換過程，轉換成簡單的路徑索引後，針對 XML 文件的 DOM 樹做全範圍的搜尋時，運用類似 Search Filter 的方法，來過濾掉一些極為可能是不必要的查詢處理程序，以此來增加查詢上的效率。

第一節　雜湊函數的編碼

雜湊(hashing or hash addressing)函數運用極為廣泛，目前市面上也有資料庫軟體是使用此項技術，如 Oracle。雜湊的技術是利用特殊設計的函數計算，而能直接由提供的關鍵值，加以計算出所需要搜尋的資料儲存位置，達到直接存取，降低時間成本。

對於雜湊方法而言，雖然提供了一項可以讓查詢更快速的技術，但仍然存在一些資料超載(overflow)的問題。就目前為止，已有許多學者已提出決解 overflow 或 bucket split 問題的新技術，如 Overflow chaining、open address 等解決方法。有些學者也提出新的 hashing 技術，如 Dynamic hashing[27]、Extendible Hashing[12]、Linear Hashing[19]等。

有些 hashing 的運用方法與其它不盡相同，像是透過 Bit Map 的概念方法，來降低因同一位置的儲存區塊資料量過多(overflow)，產生資料重新切割(split)分配，而導致的失敗查詢或效率過差的情況，如 Virtual Hashing[18]，或一些 differential file Organization 相關技術的應用的方法，

如 Bloom Filter[4]。

壹、Search Filter

Search Filter 是一種查詢過濾的概念，當查詢的關鍵值經由 hashing function 的編碼後，此關鍵值的編碼值是用來做為判斷的條件依據。如圖 15 為一個簡單的 Search Filter 範例圖，當 key 值 8888 經過特別設計的 hashing function 編碼計算後，其結果值為 88，我們因此可以先到 hash table 中查詢 88 的記憶位置，是否存在於此 hash table 之中，若存在則到 A 檔案做查詢處理，若不存在則到 B 檔案做查詢處理。運用 hashing function 與 hash table 的結合，可以做為一些查詢或處理之前的過濾器，來過濾掉一些不必要的查詢動作，減少處理時間上的浪費，達到效率的提昇。

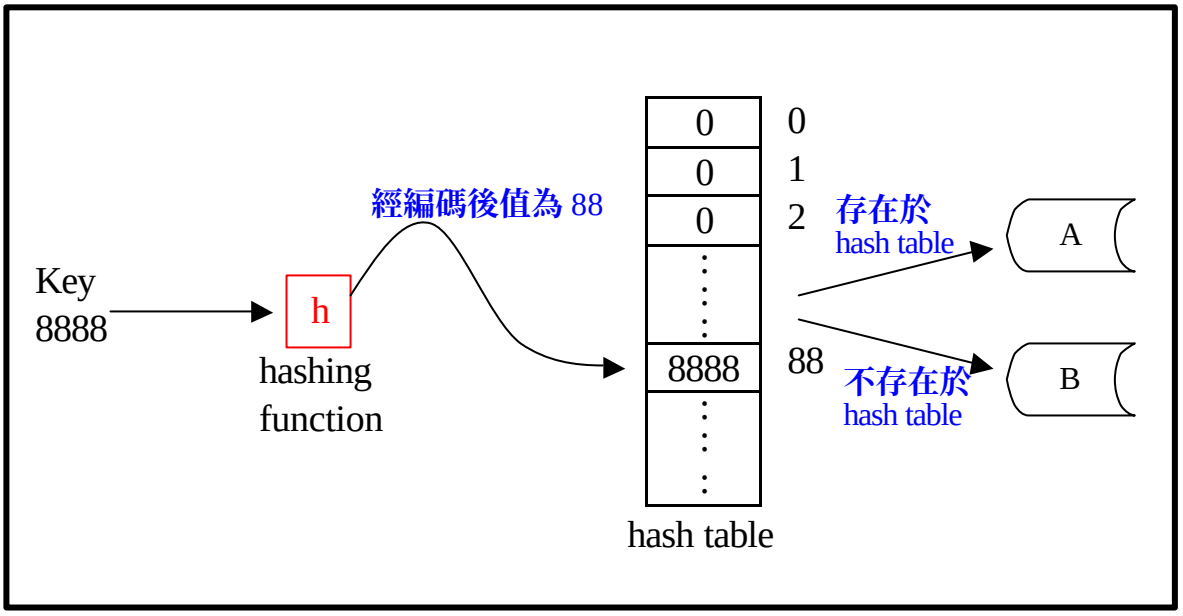


圖 15：Search Filter 圖例

貳、Bit Map

本段主要說明 Bit Map 的運用及如何將 Bit Map 使用於 Bloom Filter 的方法中。所謂 Bit Map 的概念，是運用一組相連接的記憶空

間，來做為查詢時的一個類似索引的記號，而這一組記憶空間中的每個位元(bit)都是各自獨立。各自指向不同的儲存體的記憶位置，如圖 16 為一個 Bit Map 的儲存空間，Bit Map 內的儲存值，不是 1 就是 0，1 代表 true；0 代表 false，以這種方式來做為查詢時的索引應用。

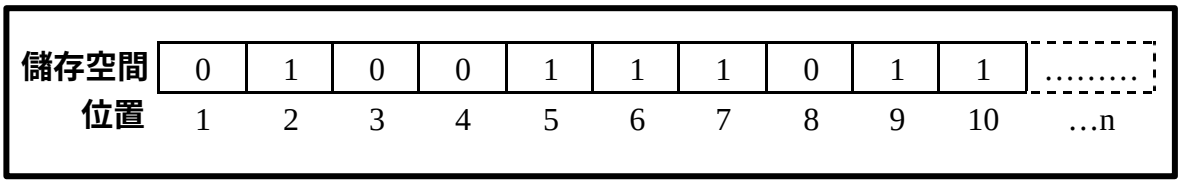


圖 16：簡單 Bit Map 範例圖

參、Bloom Filter

Bloom Filter[4]在 hash 方法上的實應性極高且效率極佳，在本篇論文所提出的方法，是運用類似 Bloom Filter 的理論方法，來改善對於 XML 的 DOM 文件樹的查詢效率，圖 17 便是一個簡單的 Bloom Filter 運用範例。

如上一節所序述，當 key 值經過 hashing function 編碼後，會先

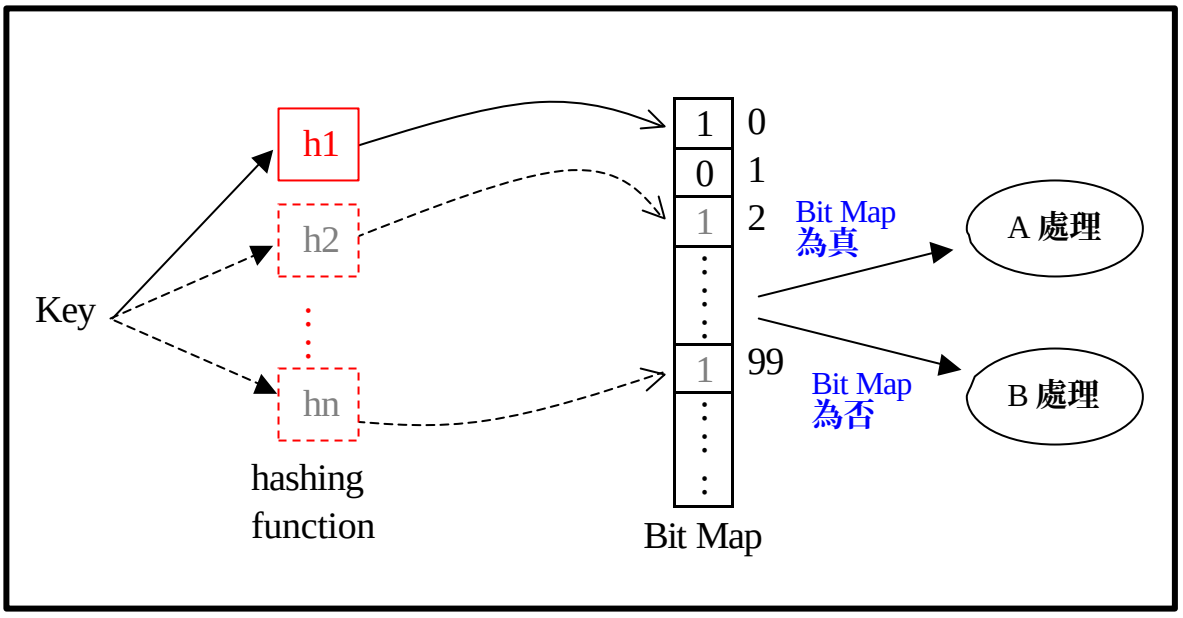


圖 17：Bloom Filter 概念應用

到 hash table 查詢是否存在，但在這裡所運用的是 Bit Map 的概念，也就是說將 hash table 縮小到只有一個 bit 的方法驗證。當然也可以運用多個不同的 hashing function 來做計算，如圖 17 所例。且將每個不同 hashing function 所計算出來的結果分別記錄於不同的位置。當查詢時也會運用這些不同的 hashing function 來加以轉換 key 值，而這些不同的 hashing function 所計算出來的結果值，再至 Bit Map 加以驗證，若皆為 true 則處理 A，反之則處理 B。

第二節 以雜湊函數為基礎的 XML 文件編碼

在本節裡主要內容是我們所提出的演算法，首先會介紹 XML 編碼方法，接著介紹此編碼方在查詢上的應用與處理，其餘部分將談論本論文所提出的方法的更進一步改良。

本論文所提出的演算法中，主要以 XML 的 DOM API 應用，將所有 XML 文件表示成為 DOM 樹，且將 DOM 樹上之每個 DOM 節點儲存成為一物件，將之儲存於物件式的資料庫中。我們以第一節所介紹之雜湊函數的概念，運用在 XML 文件樹上的編碼，做為一種查詢時過慮的方法，來增加在 XML 樹上的查詢效率。

壹、編碼方法

當有一棵 XML 的 DOM 文件樹要編碼時，首先我們會先針對此 DOM 樹上的每個 DOM 節點加以個別編碼，也就是說 DOM 樹上的每個 DOM 節點分別各自編碼，如圖 18(b)所示。當 DOM 樹上的節點，經過 hashing function 加以計算編碼後，將經過 hashing function 計算編碼後的值，對應到此 DOM 節點自已所屬的 Bit Map 中。如圖 18(a)為例，某一個元素內容為 NHU 的 DOM 節點需編碼，我們將此

DOM 節點的內容加以透過 hashing function 編碼，也就是說將 NHU 加以編碼。其 DOM 結節編碼後 $H_{(NHU)}$值為 4，也就是說 4 是代表此一 DOM 節點的位置。然後將此位置 4 對應到此 DOM 節點自已所屬的 Bit Map 中，我們便可在此 DOM 節點所屬的 Bit Map 中，編號為四的位置上，加以簽名註記(signature)為「1」。

由圖 18(b)我們可看出，Bit Map 附著於此 DOM 樹上的每個節點，且每個節點都有著自已所屬的 Bit Map 空間，且這些 Bit Map 是以位元(bit)為單位，實際上並不佔空間。

利用圖 18 的方法，將 XML 文件的 DOM 樹完全編碼，表 2 是一個透過 hashing function，將圖 13 的部分 DOM 樹編碼後的值，且

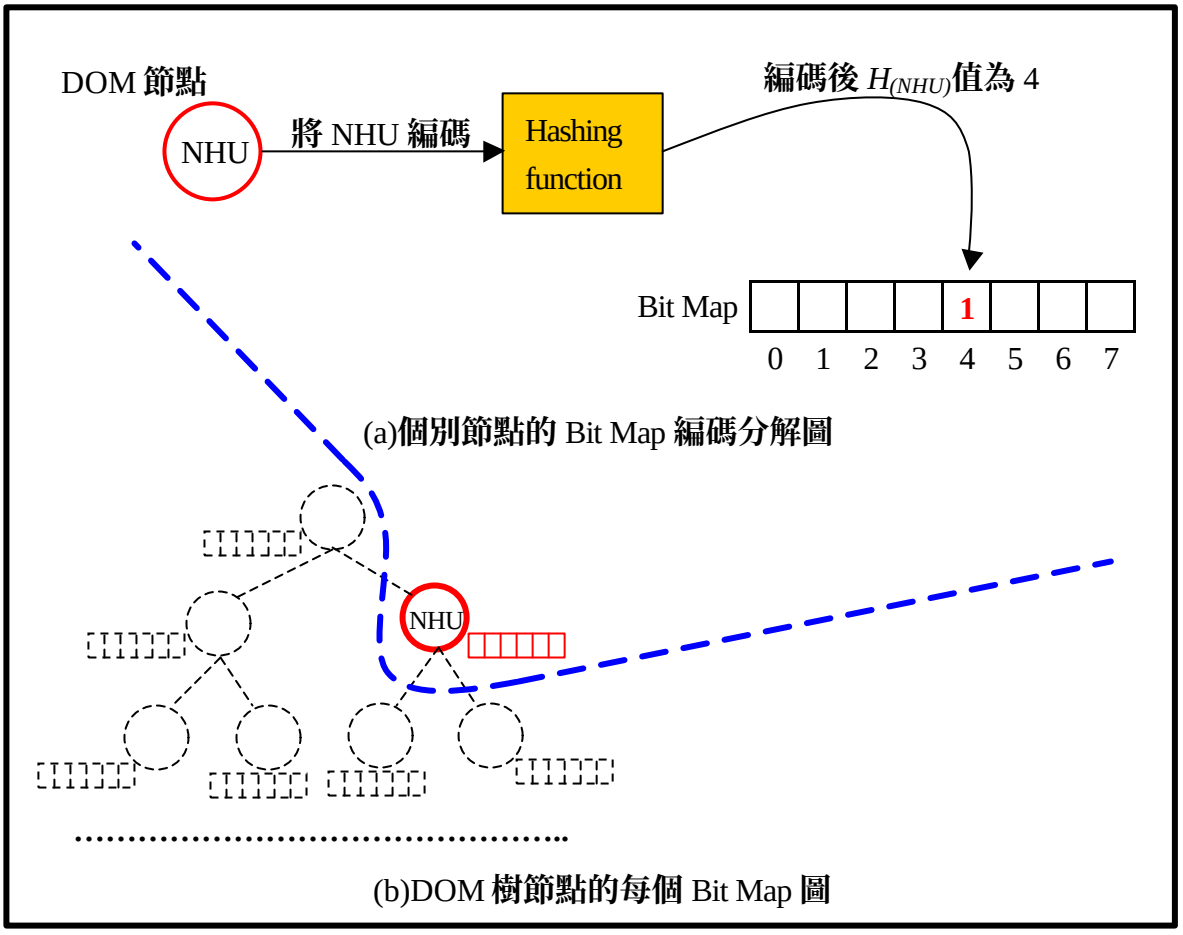


圖 18：將 DOM 樹的各別節點加以編碼

為 DOM 樹上每個別 DOM 節點的編碼值，我們將此個別節點的編碼值簡稱為 H_i（i 為 DOM 節點編號）。表 2 的第一欄為圖 13 的 DOM 節點編號，第二欄為 DOM 節點的元素(element)或屬性(attribute)內容值，第三欄為經過 8-bit 的 hashing function，將每個 DOM 元素或屬性節點內容編碼後的編碼值。

觀查表 2 的 H 編碼值，我們可以發現其編碼值與一般的編碼有所差異，在表 2 的每個 8-bit 大小的編碼值中，只有其中一個 bit 會是 1，其餘皆為 0。這代表著在這一連串的 bit 編碼中，唯一的那個

DOM 節點編碼	節點值	8-bit 編碼後值(H)
1	PERSONAL	00010000
1.1	PERSON	10000000
1.1.1	NAME	01000000
1.1.2	DEGREE	00000010
1.1.3	E-MAIL	00000001
1.1.4	TEL	00100000
1.1.1.1	Litoto	10000000
1.1.2.1	A	00000100
1.1.3.1	g0112010@….	01000000
1.1.4.1	052721001	00010000
1.2	PERSON	10000000
1.2.1	NAME	01000000
1.2.2	DEGREE	00000010
1.2.3	E-MAIL	00000001
1.2.4	TEL	00100000
1.2.1.1	C-T Li	00010000
1.2.2.1	B	00010000
1.2.3.1	litoto@……	00000001
1.2.4.1	052721002	01000000

表 2：DOM 樹的個別節點編碼 H

為 1 的 bit 位置，代表著此 DOM 節點的內容。舉個例子，如圖 15(a)所例，NHU 經過 8-bit 的 hashing function 編碼後的值為 4，4 所指的是一個位置值，也就是說在 8-bit 的空間中，4 代表著每個內容可能為 NHU 的 DOM 節點。也就是說，只要是 Bit Map 中第四個位置為內容 1，就表示可能有所要尋找的 NHU 存在。

貳、向上記錄

表 2 只是一個單純對 DOM 樹上的每個節點做分別的編碼而已，在我們的演算法裡還必需要有一個步驟，稱之為向上記錄。所謂的向上記錄意指，在每個 DOM 節點的編碼結果值，不但要儲存在 DOM 節點自已所屬的 Bit Map 以外，還必須要儲存於自已本身節點與祖先節點的 Bit Map 裡，這就是所謂向上記錄。

如圖 19 為例，節點 1.1.1 的編碼結果(Bit Map 位置 1)不但要記錄在自已所屬的 Bit Map 空間上，還必須分別記錄於節點編號為 1.1 及編號為 1 的節點上。以同樣的方法，1.1.2 編碼後的值，也必須將

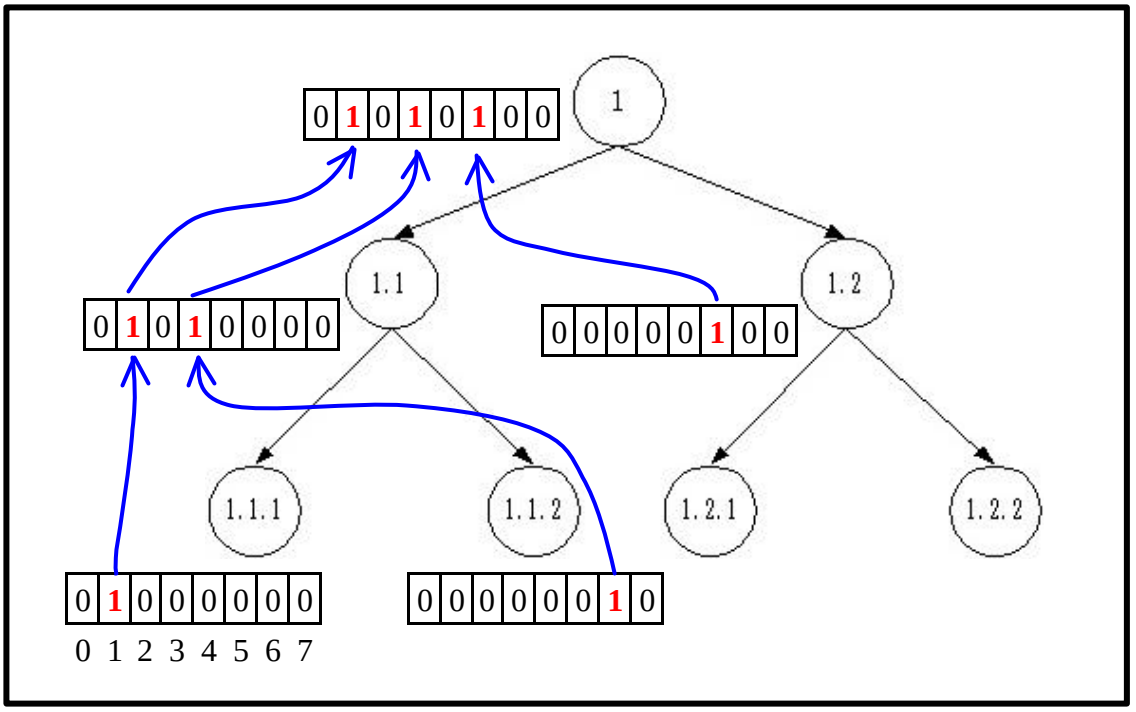


圖 19：向上記錄圖

之向上記錄於節點編號為 1.1 及編號為 1 的節點中。利用這樣的法則，將 DOM 樹上的所有節點的編碼結果，透過向上記錄的過程，將之記錄於自已本身與祖先節點的 Bit Map 空間中。完成向上記錄的過程，既代表著完成此 XML 文件的所有編碼作業，如演算法 1。

表 3 為圖 13 之部分 DOM 節點樹之向上編碼後的值，簡稱為 *S*，我們不難發現，愈是接近根節點的 Bit Map 內容中，成出現愈多的 1 值；愈是接近葉節點的 Bit Map 會出現愈少的 1，較多的 0。這便是向上記錄後的結果。由上述的方法中，我們可歸納出法則 1，如下列。

法則 1

$S_i \subset S_i'$

S 為表 2 之向上整合值

i 為 DOM 節點編號

S_i'為 S_i的子樹

圖 5 可由法則 1 可得下列結果：$S_{(1)} \subset S_{(1.1)(1.2)}$、$S_{(1.1)} \subset S_{(1.1.1)(1.1.2)(1.1.3)(1.1.4)}$、$S_{(1.2)} \subset S_{(1.2.1)(1.2.2)(1.2.3)(1.2.4)}$。

DOM 節點編碼	8-bit 向上記錄後編碼值(*S*)
1	11110111
1.1	11110111
1.1.1	11000000
1.1.2	00000110
1.1.3	01000001
1.1.4	00110000
1.2	11110011
1.2.1	01010000
1.2.2	00010010
⋮	⋮

表 3：DOM 樹向上編碼值 *S*

演算法 1

```
/*s 為儲存 hashing function 編碼之值*/

Encoding(node)
{
s=0;
    For (node.child)
        /判斷是否有子節點/
        Encoding (node.child);
    Next for
    s=s V hash (node);
    /將 node 透過 hashing function 編碼/
    /向上記錄步驟/
    sign_bitmap(node, s)
    /將編碼值記錄於附屬 DOM 結點的 Bit Map 中/
}
```

第三節　雜湊函數方法的查詢處理

XML 的雜湊函數查詢方法，主要是以上一節所介紹的編碼處理，將此編碼處理過後的 DOM 樹，加以運用於 XML 的查詢處理上。我們利用與編碼同樣的 hashing function，將轉換成 NFA 的規則路徑表示法的條件值加以編碼處理，並透過所設計的 Bit Map 方法，過濾掉一些查詢上不必要的過程。

在雜湊函數的查詢處理方法中，主要可分為三個步驟來進行。步驟一：將規則路徑表示法先轉換為 NFA 的計算。步驟二：再將 NFA 關鍵條件，加以透過 hashing function 編碼計算。步驟三：以條件的編碼結果進入實際的拜訪查詢。透過上述三個步驟，就可實際將編碼與查詢加以結合，發揮 Search Filter 的功效。接下來的部分會更深入介紹查詢的處理方法技術。

我們利用一個概念來處理查詢的過濾，首先我們利用附屬於每個

DOM 節點上的 Bit Map，當查詢的條件經過與 DOM 樹編碼時相同的 hashing function 加以計算編碼後，在拜訪 DOM 樹時，與向上記錄後的 Bit Map 加以比對，就可以很輕易的得知，查詢句所要查詢的內容條件，是否存在於目前拜訪的節點中或其子孫節點之中，如圖 20 的查詢步驟圖範例所示。假設有一個經過 NFA 轉換後的查詢式，其中一查詢關鍵條件值的內容為「NHU」，首先必須將此條件值，經過 8-bit 的 hashing function 加以編碼計算，假設 NHU 條件的編碼結果值為 7，接下來的步驟為實際拜訪查詢的步驟。編碼值為 7，7 代表 8-bit 的 Bit Map 上的一個位置。假設於目前所在的 Bit Map 上的第 7 個位置內容為 0(false)，既代表著在此節點的子孫節點中，並沒有查詢者所要查詢的資料內容。相反的，若 Bit Map 上的第 7 個位置內容值為 1(true)，代表在此節點的子孫節點或此節點本身，可能有包含著查詢者所需要的查詢資料內容。圖 20 的 DOM 查詢圖便是一個範例，當編碼值為 7 時，首先是先拜訪根節點(節點編號為 1 的 DOM 節點)，然對去核對編號為 1 之節點的 Bit Map 第 7 個位置內容，發為其 Bit Map 內容為 1(true)，我們便可往編號為 1 的節點之子孫節點，繼續拜訪。當拜訪至節點編號為 1.1 的 DOM 節點時，我們發現編號 1.1 節點的 Bit Map 內容，其第 7 個位置值並非為 1(true)而是 0(false)，我們由此可斷定，在編號為 1.1 的節點本身或其子孫節點，並沒有查詢者所需的資訊內容，便可直接放棄此子孫節點的查詢，轉往編號為 1.2 的兄第節點繼續進行查詢處理。依著上所述之步驟，便可完成查詢作業處理，如演算法 2。

由上述的查詢演算過程，我們可以定出下列之法則 2 內容，作為查詢的參考法則：

演算法 2

```
H /*DOM 個別節點編碼值  */
S /*DOM 向上記錄後編碼值*/
Node=fetch_node (DOM root)
        /* fetch by DFS */
For (the node is not final state from NFA)
      Do
           If   H node = S node ^ H node
                If   node = search_data
                     /比對查詢的資料/
                      Return (result);
                      /傳回查詢結果/
                 End if
           End if
           Node=fetch_node(node.next)
                /存取子一個節點/
      While (the node is not final state from NFA)
      Node = fetch_node (node.brother)
            /存取一個兄弟節點/
Next for
```

法則 2

假設 $S_i \subset H_i$ **則** H_i **必不存在於** S_i' **中**

S　**表** 2 **之向上記錄編碼值**

H　**表** 1 **之個別節點編碼值**

i　DOM **節點編號**

S_i'　S_i **的子樹**

若 H_i & $S_i \notin H_i$ **則** $S_i \subset H_i$

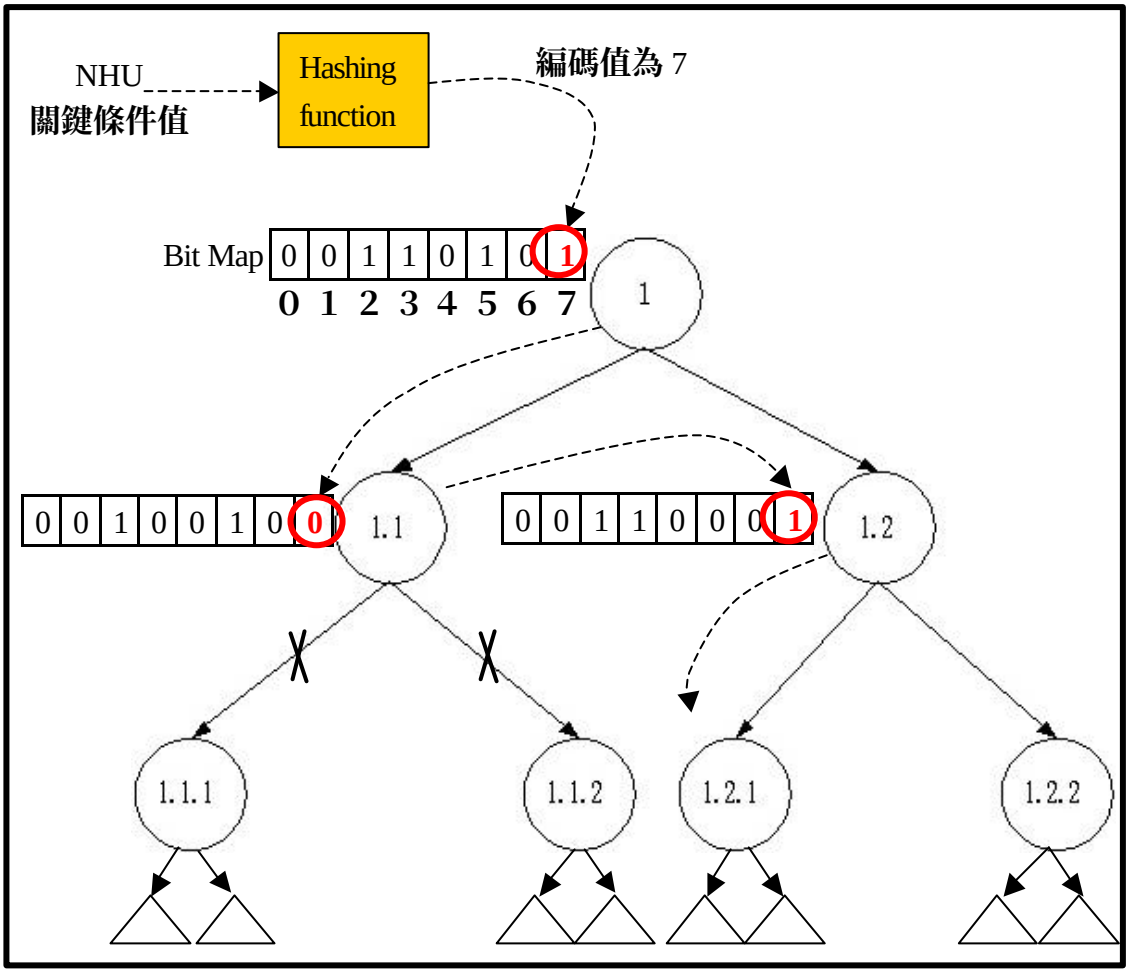


圖 20：查詢步驟圖

第四節　多元編碼與查詢

在所有的編碼方法中，都免不了會產生資料量因過多而導至的編碼重復，這種情形雖然無法避免，但卻可以將它降到最低的產生率，避免浪費過多的成本。本論文所提出的方法，可有多種不同的改善效率的方法，可避免因編碼結果相同所導至的編碼重疊[1]，且在這些方法中，並不會造成過大的負擔與成本。

壹、擴大 Bit Map 的編碼空間

[1]編碼重疊意指，不同的編碼來源值，經過同一個 hashing function 編碼後，產生相同的編碼結果值。

擴大 Bit Map 的編碼空間，也就是將編碼長度增加, 例如從 1byte 擴增至 2bytes 或更多，如圖 21(a) Bit Map 虛線部分。由於本編碼演算法只用一個位置來代表結果，這種有別於其它編碼方式的方法，並不會有最差的狀況產生(如編碼結果全部都產生 1(1111….1111)，永遠只有一個位元會是 1(000100….000)，其餘皆為 0，自然而然會因編碼長度增加，而重覆性就會降低。

貳、多元雜湊函數的編碼計算

利用此方法，最主要是可以把編碼重覆發生的可能性再降低，就可以利用多個 hashing function 來對同一個編碼來源值加以編碼，產生多個不同的編碼結果，再將這些不同的結果記錄於 Bit Map 中，如圖 21(b)。

當處理查詢時，也利用這些相同的 hashing function，來對查詢條件加以編碼，將產生的不同多個編碼值，分別到不同的 Bit Map 位值上，加以驗證如圖 22 所例。將 NHU 的查詢關鍵值，透過兩個不同的 hashing function 加以編碼，產生兩個不同的結果值，且在查詢的處理過程中，必需將此兩個不同的編碼值，都與附屬於每個 DOM 節點上的 Bit Map 加以驗證，兩者皆成立才可以繼續往節點的子孫節點查詢拜訪，若其中有一個不成立(如圖 22 之 1.1 節點的 Bit Map 圈圈部分)，則放棄此節點的子孫節點查詢，轉向此節點的兄弟節點查詢。雖然此方法必須在編碼空間較大下(一般而言是超過 2 bytes)才會有成效產生，才會有較明顯的成效率提昇，但這樣的方法可以達到多元驗證的效果，可明顯降低因編碼重覆的查詢判斷失誤，適合運用於資料量較為龐大的情況下。

參、多元儲存

此方法類似於多元雜湊函數的編碼計算，但不同的是多元儲存的方法，會將不同的 hashing function 所計算出的不同編碼值，記錄於不同 Bit Map 空間上，如圖 21(c)。在查詢處理的過程上，就如同圖 22 般，只是將不同的編碼值，分別到所屬的不同的 Bit Map 中，做查詢驗證。此方可改進利用兩個 hashing function 編碼，儲存在同一 Bit Map 中，所造成的容易填滿問題。透過將不同 hashing function 編碼所產生的結果，儲存於不同的 Bit Map 中，就可以避開此類的問題產生，雖然會使用較多的儲存空間，但卻可以改善效率。

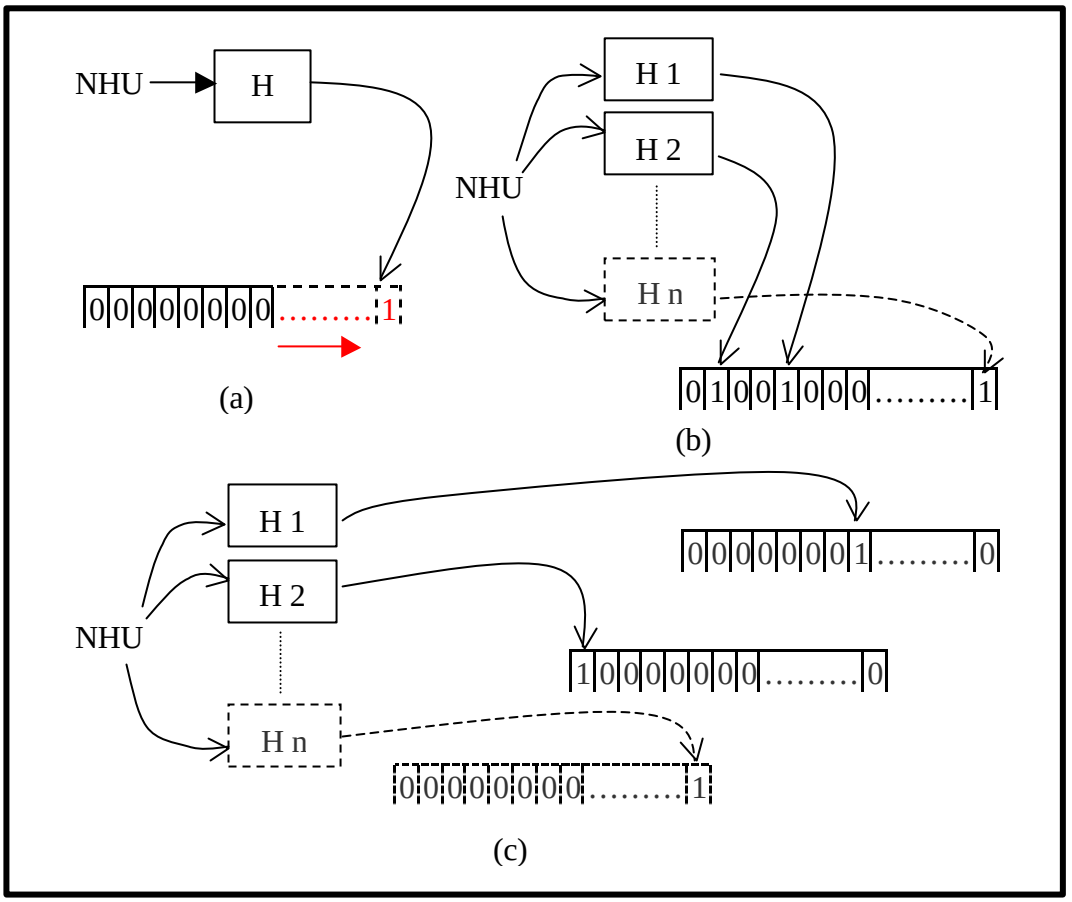


圖 21：多元編碼圖示

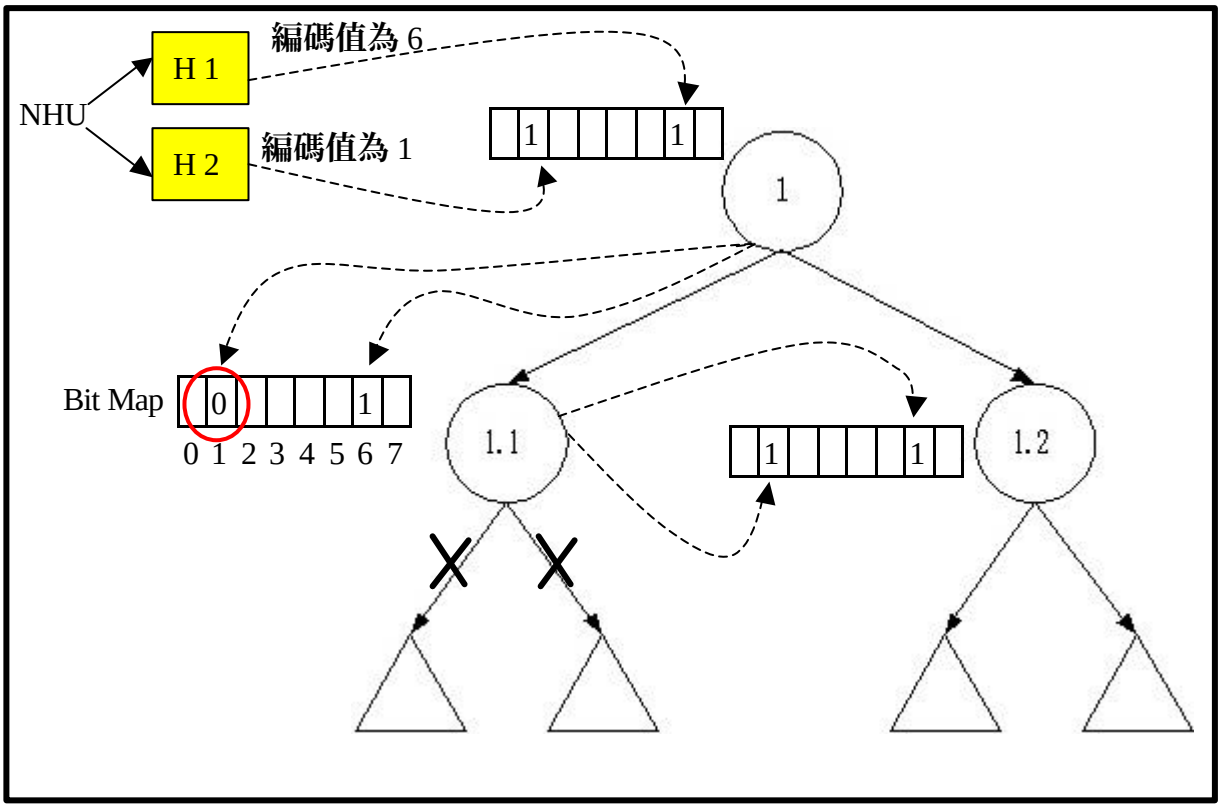


圖 22：多元雜湊函數的編碼查詢範例圖

第四章　實驗結果分析

本章節提出據具體的實驗結果分析，我們將實驗結果分為三個部分，第一部分為本論文方法與 s-DOM 方法的比較分析，第二部分是利用單一 hashing function 與兩個 hashing function 的效率比較分析，第三個實驗部分是本論文方法時間上的分析。項先必須假設本論文所使用的 XML 資料是屬異動性較小的資料內容，實驗程式撰寫採用 Java 語言，以先深後廣(DFS)的方式來進行樹的完整拜訪，並將 XML DOM 樹的所有節點儲存如物件式資料，視之為樹狀結構，並儲存於物件式資料庫中。查詢作業採用存取物件的方式，來對 XML DOM 樹上的每個節點加以進行處理。

第一節　本論文方法與 s-DOM 方法的實驗比較

本實驗利用了兩個 XML 資料，這兩個檔案名稱為 Shakespeare[24] 與 The book of Mormon[24]。我們將這些資料以 XML 來表示，共用了 4 個查詢句來對此兩個資料做查詢實驗分析，在本節的每個查詢實驗中，都會與 s-DOM 方法做比較。

圖 23 是利用表 4 的查詢子句來對 Shakespear 檔案做資料的查詢，X 軸是實驗編碼的大小，以 byte 為單位；Y 軸是查詢過程中所拜訪的節點總數。結果顯示出，本論文的方法與 s-DOM 皆於編碼大小為 2 個 bytes 時就成隱定的成長，但本論文的方法效率卻高於過 s-DOM非常多，s-DOM 所捉取的節點遠遠高出於本論文的方法。此查詢實驗，利用本論文所提的方法，存取的節點數減少了 85%。

資料庫名稱	節點個數	檔案大小	實驗查詢語句
Shakespeare	537,621	7.5Mbytes	PLAY.*[2].PERSONA

表 4：與 s-DOM 比較查詢 Shakespear 資料內容，利用 PLAY.*[2].PERSONA

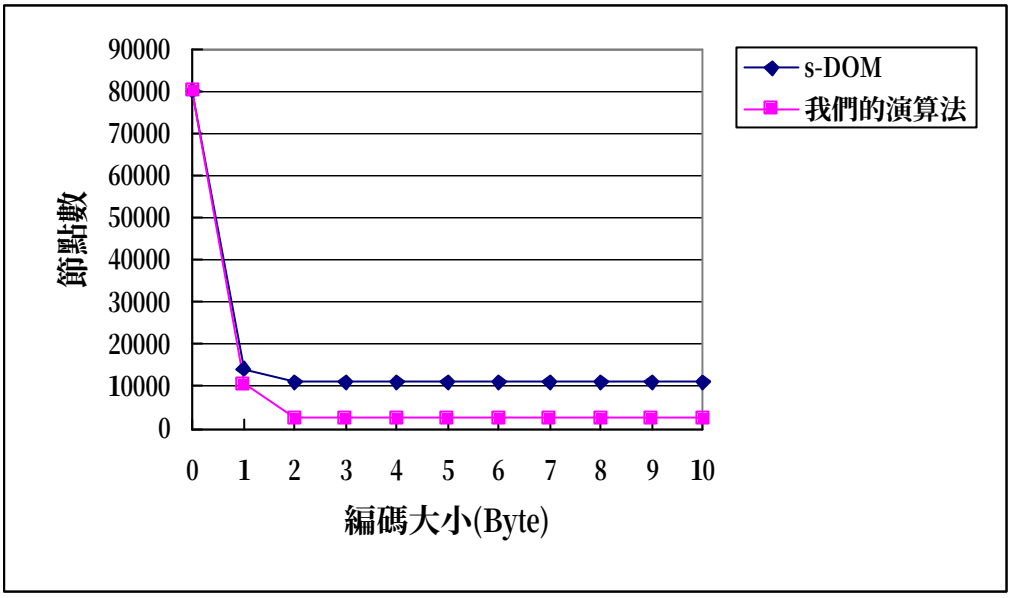


圖 23：與 s-DOM 比較查詢 Shakespear 資料內容實驗結果，利用 PLAY.*[2].PERSONA 查詢句

圖 24 之實驗結果圖，是採用表 5 的查詢子句來對 Shakespeare 檔案做查詢. 由此結果可得知，本論文所提方法與 s-DOM，在以 1 byte 為編碼大小時，本論文的方法所捉取的節點數遠遠小於 s-DOM，且於 2 bytes 為編碼大小時就趨近於隱定的成長；而 s-DOM 卻要在以 4 bytes 為編碼大小時，才能夠達到隱定的成長。在此比較實驗結果顯示，我們的方法存取的節點數減少了 50.6%的效率。

圖 25 是採用表 6 的查詢子句對檔案 The book of Mormon 做查詢，此結果顯示本論文的方法與 s-DOM 並無極大差異，不論編碼大小如何，成長皆不會影響太大，都成隱定性的成長，但是由圖可得知，s-DOM 效率雖然不至於很差，但仍舊是不如本論文的方法，在此比較實驗結果顯示，存取的節點數減少了 25.3%。

資料庫名稱	節點個數	檔案大小	實驗查詢語句
Shakespeare	537,621	7.5Mbytes	*.TITLE

表 5：與 s-DOM 比較查詢 Shakespear 資料內容，利用*.TITLE 查詢句

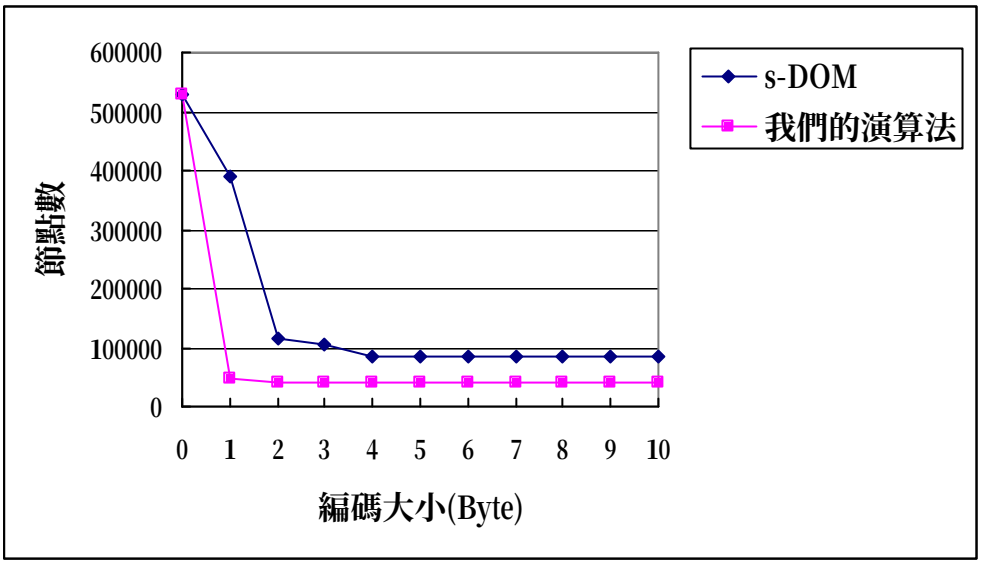


圖 24：與 s-DOM 比較查詢 Shakespear 資料內容
實驗結果，利用*.TITLE 查詢句

資料庫名稱	節點個數	檔案大小	實驗查詢語句
The book of Mormon	142,751	6.7Mbytes	tstmt.*[1].(title|ptitle)

表 6：與 s-DOM 比較查詢 The book of Mormon 資料內容，利用 tstmt.*[1].(title|ptitle)

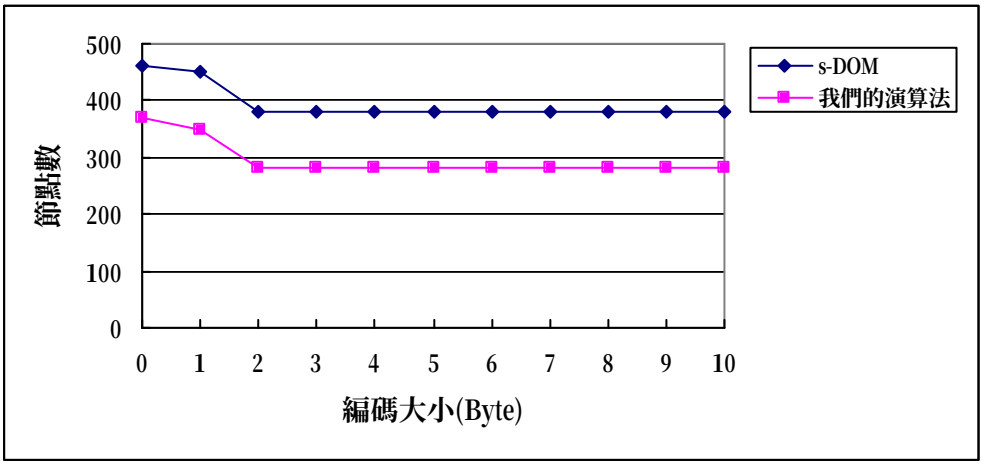


圖 25：與 s-DOM 比較查詢 The book of Mormon 資料
內容，利用 tstmt.*[1].(title|ptitle)

圖 26 是採用表 7 的查詢子句對 The book of Mormon 檔案做查詢，結果顯示本論文的方法於 2 bytes 時就成隱定成長，且平均所捉取節點數皆小於 s-DOM 約 4 萬個節點左右。此比較查詢中，存取的節點數減少了 44.2%。

資料庫名稱	節點個數	檔案大小	實驗查詢語句
The book of Mormon	142,751	6.7Mbytes	*.chapter

表 7：與 s-DOM 比較查詢 The book of Mormon 資料內容，利用*.chapter

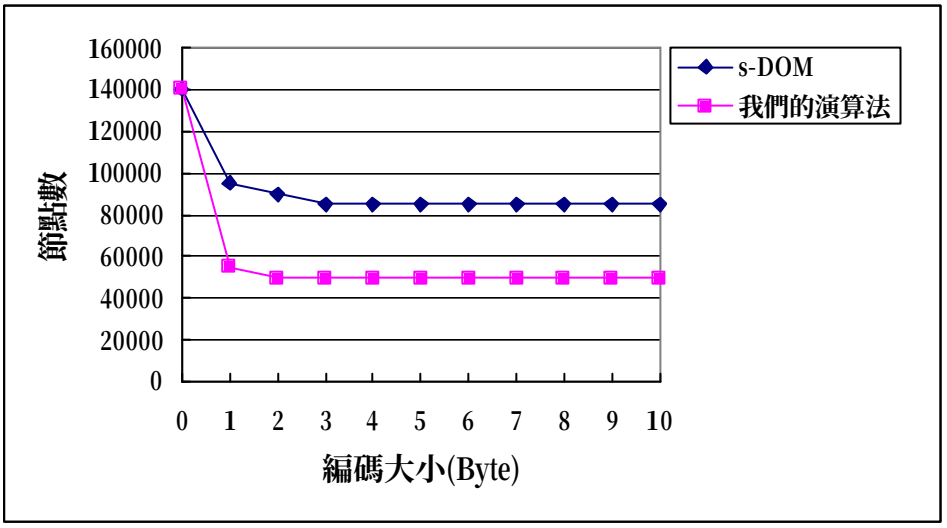


圖 26：與 s-DOM 比較查詢 The book of Mormon 資料內容，利用*.chapter

節二節　利用兩個雜湊函數的實驗分析

利用兩個不同的 hashing function，來對同樣的查詢條件值加以編碼計算與做查詢處理，此實驗結果顯示出，在利用 2 個不同的 hashing function 的方法中，解決了單一 hashing function 編碼可能造成的編碼重覆問題，改善了只利用單一 hashing function 來編碼計算的查詢效率。此節的所有實驗結果圖，階是利用單一的 hashing function 與利用兩個不同的 hashing function 的方法，來做實驗分析與比較。

圖 27 是利用兩個不同的 hashing function 來編碼， 利用表 8 的查詢子句來查詢 Shakespeare 檔案的資料，效率約可從利用單一 hashing function 的編碼方式，存取節點數可再改減少 37.9%。

資料庫名稱	節點個數	檔案大小	實驗查詢語句
Shakespeare	537,621	7.5Mbytes	PLAY.*[2].PERSONA

表 8：以兩個 hashing function 來查詢 Shakespear 資料內容，利用 PLAY.*[2].PERSONA 查詢

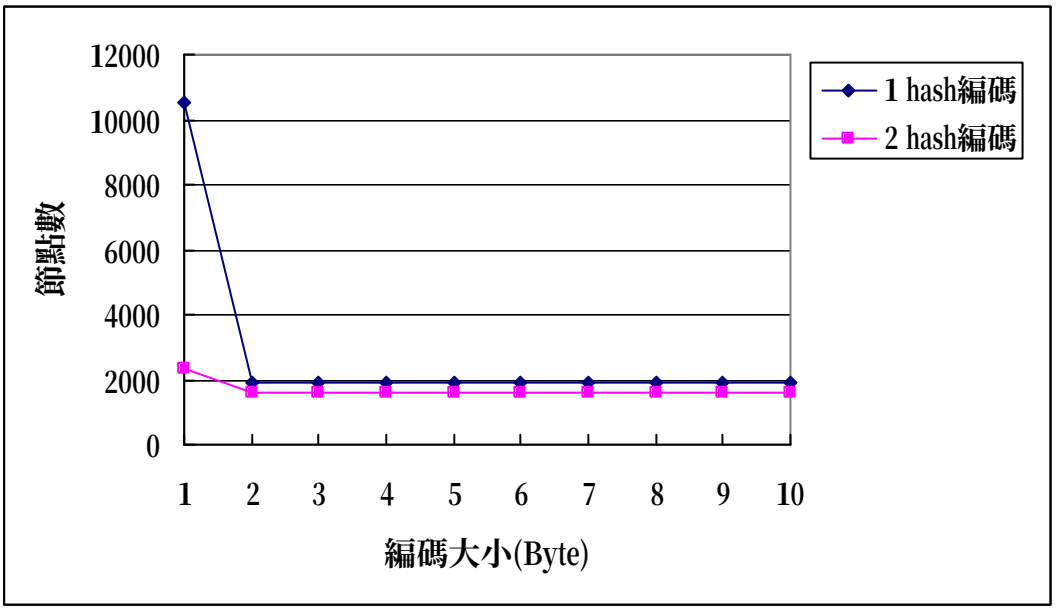


圖 27：以兩個 hashing function 來查詢 Shakespear 資料內容，利用 PLAY.*[2].PERSONA 查詢

圖 28 也是利用單一的 hashing function 與兩個不同的 hashing function 來編碼計算，而利用表 9 查詢子句來查詢的資料 Shakespeare。效率可比利用一個 hashing function 的方法，存取節點數可再改減少 33.8%。

資料庫名稱	節點個數	檔案大小	實驗查詢語句
Shakespeare	537,621	7.5Mbytes	*.TITLE

表 9：以兩個 hashing function 來查詢 Shakespear 資料內容，利用*.TITLE 查詢

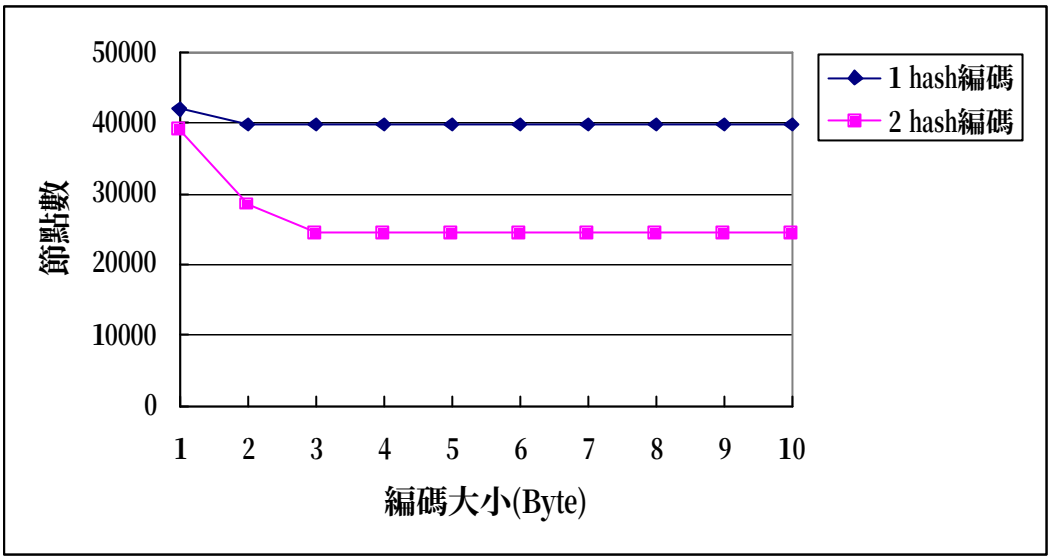


圖 28：以兩個 hashing function 來查詢 Shakespear 資料內容，利用*.TITLE 查詢

圖 29 與 30 都是利用兩個不同的 hashing function 來計算編碼，分別是利用表 10 的 tstmt.*[1].(title|ptitle)查詢句與表 11 的*.chapter 的查詢句，來查詢 The book of Mormon 的資料內容。分別可以將存取的節點數從單一的 hashing function 方式，再改減少 20.7%與 31.1%。

資料庫名稱	節點個數	檔案大小	實驗查詢語句
The book of Mormon	142,751	6.7Mbytes	tstmt.*[1].(title|ptitle)

表 10：以兩個 hashing function 來查詢 The book of Mormon 資料內容，利用 tstmt.*[1].(title|ptitle)查詢

資料庫名稱	節點個數	檔案大小	實驗查詢語句
The book of Mormon	142,751	6.7Mbytes	*.chapter

表 11：以兩個 hashing function 來查詢 The book of Mormon 資料內容，利用*.chapter 查詢

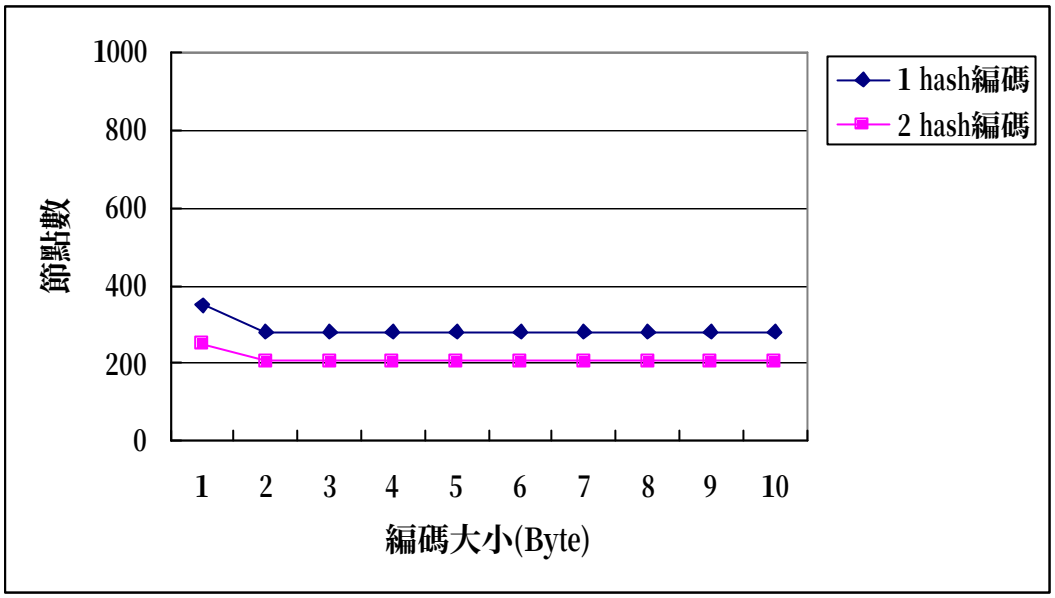


圖 29：以兩個 hashing function 來查詢 The book of Mormon 資料內容，利用 tstmt.*[1].(title|ptitle)查詢

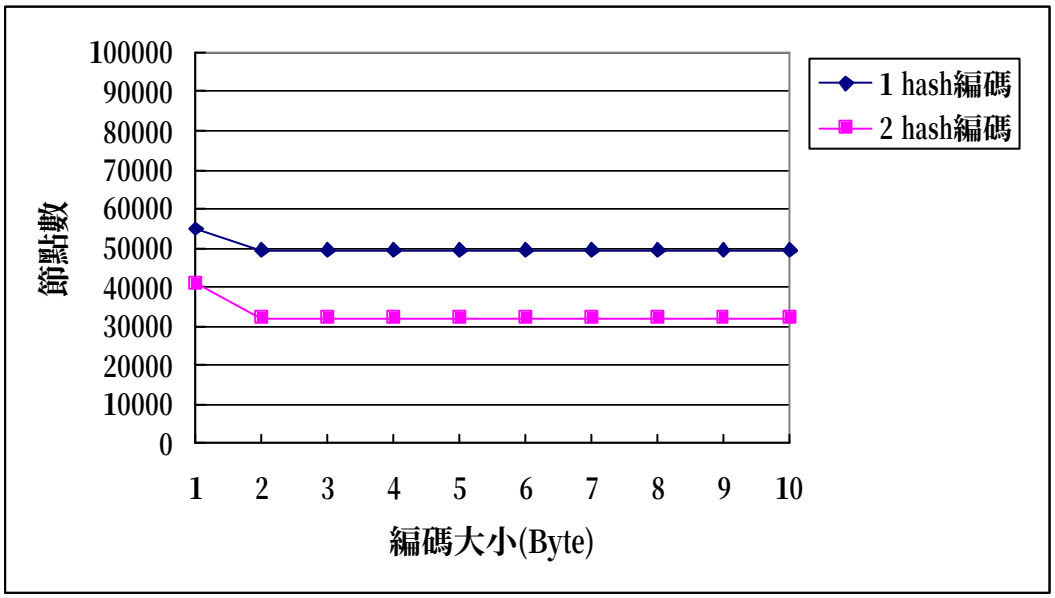


圖 30：以兩個 hashing function 來查詢 The book of Mormon 資料內容，利用*.chapter 查詢

第三節 時間相關分析

本節提出有關於時間上的實驗結果分析，本節實驗是利用表 12 的四個查詢句來做實驗估計，且將 page size 固定為 5 bytes。由圖 31 我們可發

現，在某些情況下的 page I/O 數是較大(Y 軸部分)，如查詢 2 就是個例子，原因是因為查詢 2 的查詢子句是屬於較大的範圍查詢，「*」號出現於 TITLE 之前，也就是說可能必需將大部分的子樹拜訪過，才能找出符合條件的內容。圖 32 是利用表 12 的四個查詢子句，在查詢上所花費的時間統計圖，不論編碼空間如何的變化，在時間上仍是很隱定。

	查詢資料檔案	查詢子句
查詢 1	Shakespear	PLAY.*[2].PERSONA
查詢 2	Shakespear	*.TITLE
查詢 3	The book of Mormon	Tstmt.*[1].(title|ptitle)
查詢 4	The book of Mormon	*.chapter

表 12：時間實驗分析的四個查詢句與資料檔

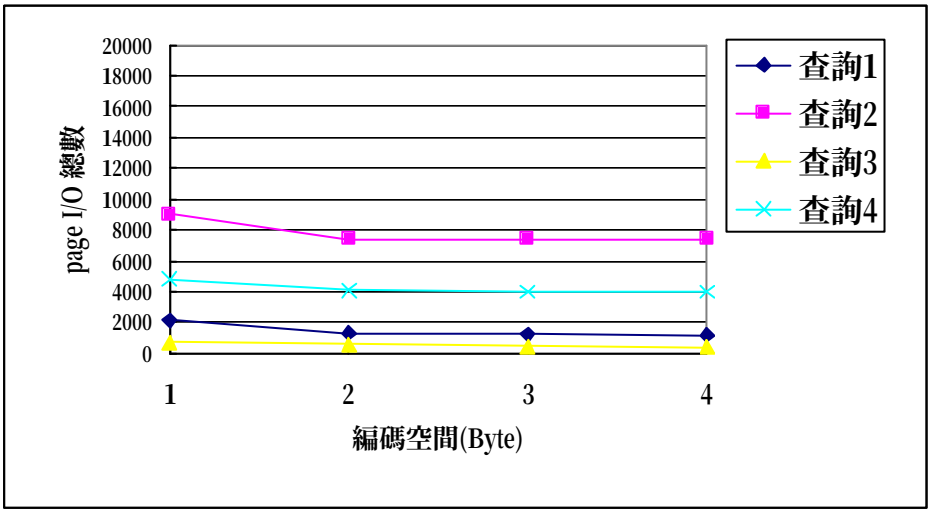


圖 31：編碼空間與 page I/O 數的實驗結果圖

由以上面的實驗中我們可得知，在本論文所提出的演算法效率是高過於 s-DOM 的方法。以穩定性的觀點來看，本論文的方法通常只要以 2 個 bytes 的編碼大小，就能夠讓結果達到一定的效率水準，且成穩定性的成長。就編碼的方面來觀查，本論文所提出的編碼方法，相較於其它類

似 ASCII 的編碼方法比較，更容易以較小的儲存空間來達到效率的提昇，是個低重疊性及高效率的方法。表 13 利用本論文所提出方法與 s-DOM 編碼方法的實驗估計比較效率提昇百分比，表 14 是利用本論文所提出之方法的 1 個 hashing function 編碼與 2 個 hashing function 編碼的比較效率提昇百分比。資料與查詢句為實驗所用的資料內容與查詢子句，中間數據則為比較後的效率提昇百分比。

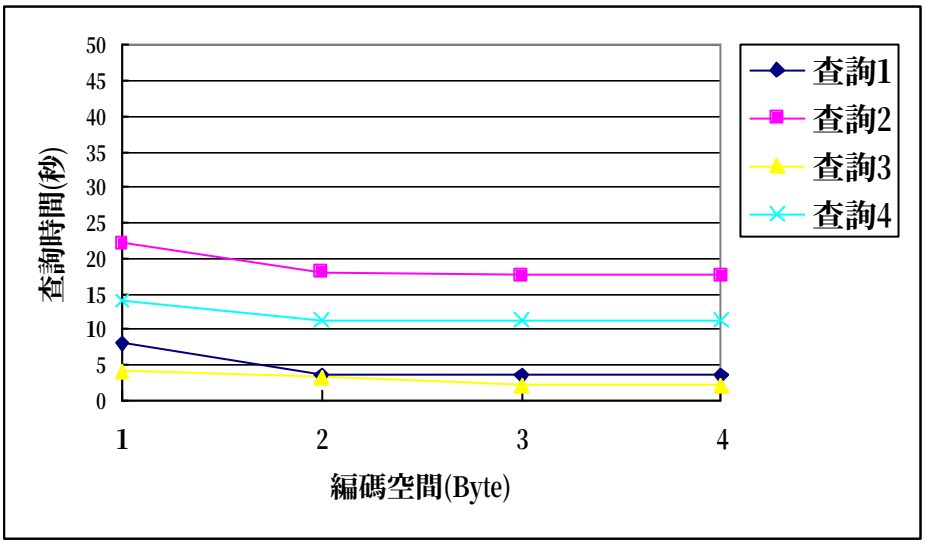


圖 32：編碼空間與查詢時間的實驗結果圖

實驗估計方法 \ 資料與查詢句	Shakespear 資料與 PLAY.*[2].PERSONA 查詢句	Shakespear 資料與 *.TITLE 查詢句	The book of Mormon 資料與 stmt.*[1].(titlelptitle) 查詢句	The book of Mormon 資料與 *.chapter 查詢句
本論文方法與 s-DOM 方法的比較	85%	50.6%	25.3%	44.2%

表 13：本論文方法與 s-DOM 方法比較效率提昇百分比表

實驗估計方法 \ 資料與查詢句	Shakespear 資料與 PLAY.*[2].PERSONA 查詢句	Shakespear 資料與 *.TITLE 查詢句	The book of Mormon 資料與 stmt.*[1].(title|ptitle) 查詢句	The book of Mormon 資料與 *.chapter 查詢句
利用本論文提出方法的 1 個 hashing function 與 2 個 hashing function 的方法相比較	30.8%	37.9%	33.8%	20.7%

表 14：利用本論文方法的 1 個 hashing function 編碼和 2 個 hashing functio 編碼比較的效率提昇百分比表

第五章　結論與建議

本篇論文所提出的編碼方法中，不但與以往的傳統編碼方式有所差異，在查詢的處理過程中，更可排除一些傳統編碼上的最差狀況，讓編碼的結果不會因為資料量的大小，影響查詢時的效率。尤其在傳統的索引(index)無法有效的被使用時，本論文所提出的方法，效果更是彰顯。

在本論文中，使用了以雜湊函數為基礎的編碼註記查詢方法，利用同樣的 hashing function 來做文件編碼與查詢的編碼，讓在查詢擷取節點時，能以過濾的方法，來檢驗我們所要查詢的條件內容，是否存在於目前節點下的子樹之中，此方法不但可以大大的裁減掉不必要的拜訪過程，且決解了 s-DOM 編碼方式上的缺點，不但可以提昇查詢的效率也下降了查詢的時間。

XML 為當今興新的研究方向，在查詢的技術上更是一項很大的研究空間，目前為止仍無一定的標準，眾說紛紜，且本論文採 W3C 所提之規則路徑表示法，再透過本論文所提出之編碼，方法來達到最佳化的查詢，在效率上也具有高度的發展空間。實驗結果顯示，我們所提的方法比其它方法來的快速且有效。

在往後的 XML 研究方向裡，在 NXD 的查詢方法與技術上的研究還相當的缺乏，相信在這方向的查詢方法與研究，會是個相當大的研究領域空間。

文獻參考

中文文獻參考：

[1] 吳柏璋，「XML 文件儲存方式之研究」，朝陽科技大學資訊管理研究所碩士論文，91 年 7 月。

[2] 陳長念、陳勤意，網頁新視界 XML 入門與應用，松崗電腦圖書資料設份有限公司，台北，民國八十八年。

英文文獻參考：

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom and J. Wiener, "The Lorel Query Language for Semistructured Data," International Journal on Digital Libraries, Vol. 1, No. 1, pp. 68-88, 1997

[2] S. AlKhalifa, H. V. Jagadish, N. Koudas, Jignesh M. Patel, D. Srivastava and Y. Wu, "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," 18th International Conference on Data Engineering, February 26 - March 01, 2002, San Jose, California.

[3] A. Bonifati and S. Ceri, "Comparative Analysis of Five XML Query Languages," ACM SIGMOD Record , 29(1), 2000.

[4] B. H. Bloom, "Space/time trade offs in hash coding with allowable errors," Communications of the ACM, 13(7), pp. 422-426, July 1970.

[5] E. Bertino and B. Catania, "Integrating XML and Database," IEEE Internet computing, Vol. 5 Issue: 4, July-Aug, pp. 84-88, 2001.

[6] P. Buneman, S. Davidson, G. Hillebrand and D. suciu. "A Query Language and Optimization Techniques for Unstructured Data," SIGMOD, 1996.

[7] Timo Bohme and Erhard Rahm, “Xmach-1: A Benchmark for XML Data Management,” Proceedings of German database conference BTW2001, Oldenburg, 7-9. March, Springer, Berlin, 2001.

[8] V. Christophides, S. Abiteboul, S. Cluet and M. Scholl, “From Structured documents to Novel Query Facilities,” SIGMOD, 1994.

[9] eXcelon, “An XML Data Server For Building Enterprise Webapplications,” http://ww.odi.com/products/white-papers.htm.

[10] D. Florescu and D. Dossmann, “Storing and Querying XML data using an RDBMS,” IEEE Data Engineering Bulletin, Vol. 22, No. 3, pp. 27-34, 1999.

[11] D. Florescu and D. Kossman, “Aperformance Evaluation of Alternative Mapping Schemes for Storing XML Data in Relational Database,” Rapport de Recherche No. 3680 Inria, Rocquencourt, France, May 1999.

[12] R. Fagin, J. Nievergelt, N. Pippenger and H. R. Strong, “Extendible Hashing-A Fast Access Method for Dynamic Files,” ACM Trans. Database System 4, pp. 315-344, Sept. 1979.

[13] S. Flesca, F. Furfaro and S. Greco, “XGL: a graphical query language for XML,” International Database Engineering and Applications Symposium, July 17 - 19, Edmonton, Canada, 2002.

[14] For technical reports, working drafts, recommendations and specifications of XML and related technologies, see the World Wide Web Consortium(W3C)Web, http://www.w3c.org.

[15] R. Goldman and J. Widom, “DataGuides: Enabling query formulation and Optimization in Semistructured Database,” Proc. of the 23th VLDB

conference, pp. 436-455, 1997.

[16] H. Jiang, H. Lu, W. Wang and J. Xu Yu, "Path MaterializationRevisited: An Efficient Storage Model for XML Data," 2nd Australian Institute of Computer Ethics Conference (AICE2000), Canberra.

[17] P. Linz, An Introduction to Formal Languages and Automata, Houghton Mifflin, Boston, MA. 1990.

[18] W. Litwin, "Virtual Hashing: A Dynamically Changing Hashing," Proc. 4th Conf. on Large Data Bases, West Berlin, pp. 517-523, Sept. 1978.

[19] P. A. Larson, "Linear Hashing with partial Expansions," Proc. 6th Conference on VLDB, Montreal, pp. 224-232 , Oct. 1980.

[20] J. Mchugh, J. Widom, S. Abiteboul, Q. Luo and A. Ragaraman, "IndexingSemistructured Data," Technical Report, Stanford University, Computer Science Department, pp. 1-21, 1998.

[21] D. Martin, M. Birbeck, M. kay, B. Loesgen, J. Pinnock, S. Livingstone, P. Stark, K. Williams, R. Anderson, S. Mohr, D. Baliles, B. Peat and N. Ozu, Professional XML, Wrox Press Ltd, USA, March 2000.

[22] Brett McLaughlin, JavaTM and XML, O'Reilly and Associates Inc., USA, 2001.

[23] T. Milo and D. Suciu, "index Structures for Path Expressions," Proc. of the 7th International Conference on Database Theory (ICDT), pp. 277-295, 1999.

[24] S. Park and H. J. Kim, "A new query processing technique for XML based on signature," Database Systems for Advanced Applications, 2001. Proceedings. Seventh International Conference on , pp. 22 –29, 2001.

[25] S. Park and H. J. Kim, "SigDAQ: an enhanced XML query optimization technique," Journal of Systems and Software Vol. 61, Issue: 2, pp. 91-103, March 15, 2002.

[26] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. De-witt and J. Naughton, "Relational databases for querying XML documents: limitations and opportunities," In Proceedings for VLDB, Edinburgh, UK, pp. 302-312, September, 1999.

[27] M. Scholl, "New file organizations based on dynamic hashing," ACM Trans. Database Syst., 6, #1, pp. 194-211, March 1981.

[28] SAX, http://www.saxproject.org/

[29] K. Thomposn, "Regular Expression Search Algorithm," Communications of the ACM (CACM), pp. 419-422, Vol. 11, 1968.

[30] V. Tseng and W. Lin, "A New Method for Indexing XML Documents," Proc. of the 12th workshop on Object-Oriented Technology and Applications, pp. 39-46, 2001.

[31] W3C Document Object Model, http://www.w3.org/DOM/.

[32] W3C DOM Level 1, http://www,23.org/TR/REC-DOM-Level-1/.

[33] W3C DOM Level 1, http://www,23.org/TR/REC-DOM-Level-2/.

[34] XML 1.0, http://www.w3.org/TR/2000/REC-xml-20001006.

[35] XML Path Language, http://www.w3.org/TR/xpath.

[36] XML Path Language, http://www.w3.org/TR/Xquery.

[37] A. Zisman, "An overview of XML," Computing and Control Engineering Journal, August 2000.